NETWORK INTRUSION DETECTION SYSTEM WITH INCREMENTAL ACTIVE LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MÜNTEHA NUR BEDİR TÜZÜN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2022

Approval of the thesis:

NETWORK INTRUSION DETECTION SYSTEM WITH INCREMENTAL ACTIVE LEARNING

submitted by **MÜNTEHA NUR BEDİR TÜZÜN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Pelin Angın
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Pınar Karagöz
Computer Engineering, METU _____

Assoc. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assoc. Prof. Dr. Ahmet Burak Can
Computer Engineering, Hacettepe University _____

Date: 14.09.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Münteha Nur Bedir Tüzün

Signature :

ABSTRACT

NETWORK INTRUSION DETECTION SYSTEM WITH INCREMENTAL ACTIVE LEARNING

Bedir Tüzün, Münteha Nur
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Pelin Angın

September 2022, 79 pages

While Internet usage has increased every year, it has gained momentum in recent years with the global pandemic. Increasing Internet usage has brought increasing cyber threats. Intrusion detection systems have become more important than ever. The performance of these systems is directly proportional to their adaptiveness to the rapid changes in attack types. However, desired performance cannot always be achieved due to the lack of labeled data on newly developed attacks and the difficulty of incremental learning with machine learning methods. In this study, we proposed a network intrusion detection system using active learning methods for class incremental learning, which can adapt to the dynamic environment and provide high performance with less labeled data. Experiment results show that the proposed method requires fewer labeled training data instances and learns new types of attacks incrementally.

Keywords: network intrusion detection system, active learning, incremental learning

ÖZ

ARTIMLI AKTİF ÖĞRENME İLE AĞ SALDIRI TESPİTİ SİSTEMİ

Bedir Tüzün, Münteha Nur
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Pelin Angın

Eylül 2022 , 79 sayfa

İnternet kullanımı her geçen yıl artarken geçtiğimiz yıllarda global pandemi ile birlikte ivme kazanmıştır. Artan internet kullanımı siber tehditlerin artışını da beraberinde getirmiştir. Saldırı tespiti sistemleri her zamankinden daha önemli hale gelmiştir. Bu sistemlerin başarımı hızla değişen saldırı tiplerine göre kendilerini güncellemeleriyle doğru orantılıdır. Ancak, yeni geliştirilen saldırılara ait yeterli etiketli veri bulunmaması ve makine öğrenmesi metodları ile artımlı öğrenmenin zorluğu gibi sebeplerle istenen performans her zaman elde edilememektedir. Bu çalışmada, aktif öğrenme yöntemleri sınıf artımlı öğrenme için kullanılarak dinamik ortama uyum sağlayan ve daha az veri ile yüksek başarım sağlayan bir ağ saldırı tespit sistemi geliştirilmiştir. Deney sonuçları, önerilen yöntemin daha az etiketli eğitim verisi örneği gerektirdiğini ve yeni tür saldırıları aşamalı olarak öğrendiğini göstermektedir.

Anahtar Kelimeler: ağ saldırı tespiti sistemi, aktif öğrenme, artımlı öğrenme

To my love, sweetheart, dear life partner, my husband İsmail ...

ACKNOWLEDGMENTS

First of all, I want to thank my husband, İsmail Tüzün, for his support, help, and endless love. Without him, I cannot imagine how to deal with the difficulties I faced during my thesis work. Also, I want to thank my thesis advisor, Pelin Angın for her encouragement and assistance. I think she is the most helpful thesis advisor ever. I would like to thank the jury members, Pınar Karagöz and Ahmet Burak Can, for their contributions. Lastly, I thank my sister, Fatma Nur Demir, and all members of my family for their support.

The numerical calculations reported in this thesis were fully performed using the TÜBITAK ULAKBİM, High Performance and Grid Computing Center (TRUBA) resources.

This research has been supported by TÜBİTAK under grant number 120E537. The entire responsibility of the thesis belongs to the author of the thesis. The support received from TÜBİTAK does not mean that the content of the publication is approved in a scientific sense by TÜBİTAK.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vi

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xi

LIST OF FIGURES xiii

LIST OF ABBREVIATIONS xv

CHAPTERS

1 INTRODUCTION 1

2 BACKGROUND INFORMATION 5

2.1 Intrusion Detection System (IDS) 5

2.1.1 What is IDS? 5

2.1.2 Brief History of IDS 5

2.1.3 Taxonomy of IDS 6

2.2 Deep Learning 7

2.3 Active Learning 9

2.3.1 What is Active Learning? 9

2.3.2 Query Strategies 11

2.4 Incremental Learning 12
3 RELATED WORK 15
3.1 IDS with Deep Learning 15
3.2 IDS with Active Learning 17
3.3 IDS with Incremental Learning 22
4 METHODOLOGY 25
4.1 Architecture and General Properties of Proposed IDS 25
4.2 Classification Module 28
4.3 Incremental Learner Module 28
4.4 Active Learning Module 29
4.5 Simulated Oracle Module 30
5 EXPERIMENTS AND DISCUSSION 31
5.1 Dataset 31
5.2 Experimental Setup 32
5.3 Evaluation Measure 33
5.4 Results and Discussion 34
5.4.1 Least-confident Method Results 35
5.4.2 Margin Strategy Results 46
5.4.3 Entropy Strategy Results 58
5.4.4 General Discussion 70
6 CONCLUSION 73
REFERENCES 75

LIST OF TABLES

TABLES

Table 5.1 Least-confident method's general performance table 36

Table 5.2 Least-confident method's f1-score table for each class 37

Table 5.3 Least-confident method's input data and unlabeled data pool table . 37

Table 5.4 Least-confident method's unlabeled data pool and selected data table 38

Table 5.5 Least-confident method's train data table 39

Table 5.6 Margin method's general performance table 47

Table 5.7 Margin method's f1-score table for each class 48

Table 5.8 Margin method's input data and unlabeled data pool table 49

Table 5.9 Margin method's unlabeled data pool and selected data table 50

Table 5.10 Margin method's train data table 51

Table 5.11 Entropy method's general performance table 59

Table 5.12 Entropy method's f1-score table for each class 60

Table 5.13 Entropy method's input data and unlabeled data pool table 61

Table 5.14 Entropy method's unlabeled data pool and selected data table 62

Table 5.15 Entropy method's train data table 63

Table 5.16 All methods general performance comparison table 70

Table 5.17 All methods f1-score comparison table for each class 71

Table 5.18 All methods train data comparison table 71

LIST OF FIGURES

FIGURES

Figure 1.1 Total malware in the world 1

Figure 2.1 Classification of IDSs 6

Figure 2.2 Biological neuron 8

Figure 2.3 Perceptron (Artificial neuron) 8

Figure 2.4 Neural network structure 9

Figure 2.5 Active learning cycle 10

Figure 2.6 Incremental learning process 13

Figure 3.1 Categorization of deep learning methods in IDS 16

Figure 4.1 The architecture of the proposed IDS 27

Figure 5.1 Evaluation graph of least-confident method 35

Figure 5.2 Effect of initial split ratio on least confident method 40

Figure 5.3 Effect of batch size on least confident method 41

Figure 5.4 Effect of epoch size on least confident method 42

Figure 5.5 Effect of selection parameter on least confident method 43

Figure 5.6 Effect of pool size on least confident method 44

Figure 5.7 Effect of new class min count on least confident method 45
Figure 5.8 Evaluation graph of margin method 46
Figure 5.9 Effect of initial split ratio on margin method 52
Figure 5.10 Effect of batch size on margin method 53
Figure 5.11 Effect of epoch size on margin method 54
Figure 5.12 Effect of selection parameter on margin method 55
Figure 5.13 Effect of pool size on margin method 56
Figure 5.14 Effect of new class min count on margin method 57
Figure 5.15 Evaluation graph of entropy method 58
Figure 5.16 Effect of initial split ratio on entropy method 64
Figure 5.17 Effect of batch size on entropy method 65
Figure 5.18 Effect of epoch size on entropy method 66
Figure 5.19 Effect of selection parameter on entropy method 67
Figure 5.20 Effect of pool size on entropy method 68
Figure 5.21 Effect of new class min count on entropy method 69
Figure 5.22 Comparison of all methods 70

LIST OF ABBREVIATIONS

IDS	Intrusion Detection System
NIDS	Network based Intrusion Detection System
HIDS	Host based Intrusion Detection System
DL	Deep Learning
AL	Active Learning
IL	Incremental Learning
ANN	Artificial Neural Network
KNN	K-Nearest Neighbor
SVM	Support Vector Machine

CHAPTER 1

INTRODUCTION

Usage, size, and complexity of computer networks increased significantly in the last few years. Furthermore, the global COVID-19 pandemic has accelerated this increase. With remote work and education, more and more people need to use the Internet and save their data on the cloud. People are required to control a lot of things remotely by using IoT for their safety. With increased network usage, cyber threats also increase. *Figure 1.1* shows that total malware in the world has increased rapidly in the last 3 years according to the McAfee Labs Threat Report 2021 [1]. Therefore, network security has become a more vital issue than ever before.

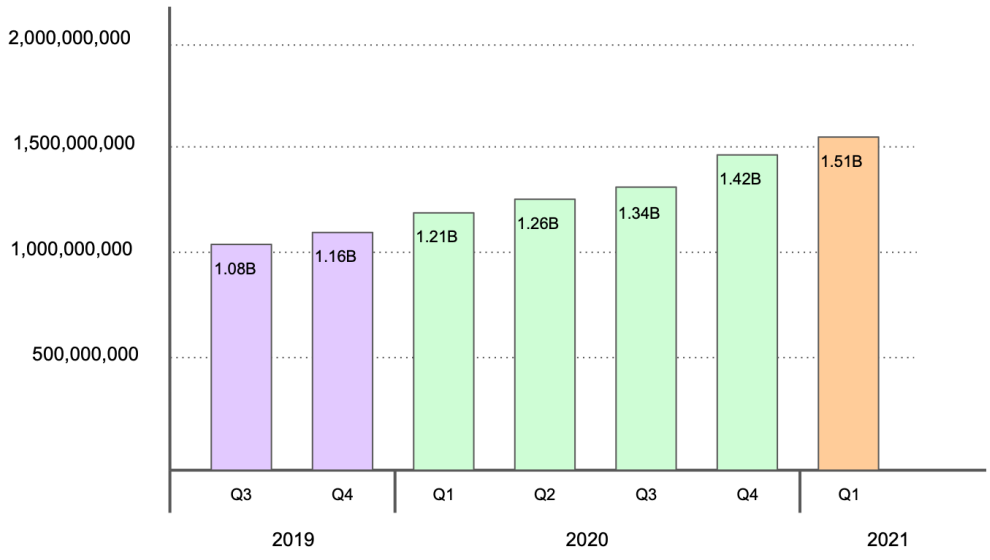


Figure 1.1: Total malware in the world

The first line of defense is that well-designed systems have standard security policies. However, a system with zero vulnerability cannot be achieved. Intrusion detection systems (IDS) are generally called the second line of defense. They are designed to detect attacks on networks or computers and alert system operators. There are two types of IDS according to deployment location: network based IDS (NIDS) and host based IDS (HIDS). NIDS analyzes network traffic data and HIDS examines processes, logs, etc. on a computer. IDSs are divided into two categories based on the analysis method they use: anomaly-based and signature-based. An anomaly-based IDS learns the normal behavior of the system and any deviation from the normal is considered an attack. On the other hand, a signature-based IDS learns known attack patterns and classifies attacks based on similarity to those patterns. Signature-based IDSs have high accuracy on known attacks, but cannot detect unknown attacks.

In today's world, lots of new attack types are developed in a short time. Therefore, an IDS must be adaptive to the highly dynamic environment. Usage of deep learning and machine learning methods in the IDS domain is very popular these days and they have a satisfying performance. Unfortunately, those methods cannot learn sequentially to adapt to changes. Researchers have been working on the lifelong learning problem for the last two decades, but it still has not been solved completely. The main challenge in incremental learning is catastrophic forgetting. It means newly learned information overwrites old knowledge learned before. Therefore, the model needs to be retrained from scratch every time new information is available. Some incremental learning methods have been developed to overcome this problem, but none of the methods provides the same performance as retraining with all training data. Thus, keeping the training data small makes the continuous learning process more efficient. In fact, there is labeled data shortage in the intrusion detection domain. This is another obstacle for obtaining high-performance IDSs. Active learning has the potential to be a solution to those problems. The active learner selects the most informative instances from the unlabeled dataset. Then, it queries selected instances to the oracle for labeling. Same and even higher performance can be achieved with less training data using active learning.

In this thesis, we proposed a signature-based network intrusion detection system with active learning. The system can learn new attack types incrementally in an efficient way. It requires fewer labeled data instances and has a short training time since the training data set is as small as possible.

The contributions of this thesis are as follows:

- All of the uncertainty strategies of active learning are applied to the intrusion detection domain, and their performances are compared.
- Parameters that affect performance while using active learning for intrusion detection are analyzed comprehensively.
- As far as we know, this is the first time that active learning is applied to the multi-class classification problem of intrusion detection domain for class incremental learning purposes.

Background information will be given about intrusion detection systems, active learning, and incremental learning in chapter 2. In the next chapter, related works will be summarized about the usage of deep learning, active learning, and incremental learning methods in the intrusion detection domain. The methodology proposed in this thesis will be explained in detail in chapter 4. Then, the results of the experiments will be shown and analyzed in chapter 5. Finally, this study will be concluded in chapter 6.

CHAPTER 2

BACKGROUND INFORMATION

2.1 Intrusion Detection System (IDS)

2.1.1 What is IDS?

Definition of intrusion detection system from Internet Security Glossary is as follows [2]:

A security service that monitors and analyzes system events for the purpose of finding, and providing real-time or near real-time warning of, attempts to access system resources in an unauthorized manner.

2.1.2 Brief History of IDS

Originally, intrusion detection was performed by system administrators monitoring user activities in front of a console. This early form of intrusion detection strategy was effective enough at that time, but it was not scalable. In the late '70s and early '80s, administrators typically printed audit logs to review for detecting unusual or malicious behavior evidence. This was the next step of intrusion detection evolution. In 1980, the first intrusion detection system was developed by James P. Anderson [3]. However, the IDS was usually executed at night since the analysis was slow. Because of that, most intrusions were detected after their occurrence. Researchers were able to develop real-time IDSs in the early '90s. This enabled real-time response for detected attacks [4]. When IDSs are evolving rapidly, attacks are also evolving and more sophisticated attacks are developed. Moreover, data volume generated and transmitted

over the Internet is increasing each year. Therefore, the performance of IDSs have become more critical for us these days. Lots of research has been conducted on intrusion detection systems, but important issues still exist. Intrusion detection systems which are more precise, capable of detecting a wide range of attacks, and lower false alarm rates are needed in today's world [5].

2.1.3 Taxonomy of IDS

IDSs are classified in two ways according to the data analysis method or deployment location as shown in *Figure 2.1*.

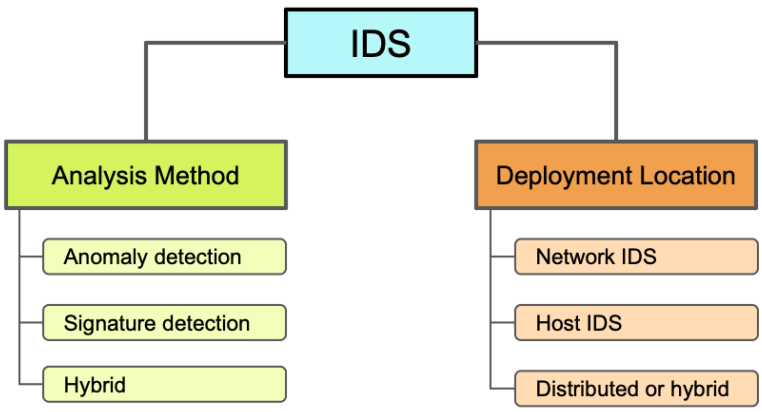


Figure 2.1: Classification of IDSs

1. **IDS Types based on Analysis Method**

   - **Anomaly detection:** Compares current behavior with the behavior of legitimate users, and deviation from the normal behavior is considered as intrusion. The advantage of this method is that it can detect unknown attacks. The disadvantage is high false positive alarm rate. [5]
   - **Signature detection:** Compares current behavior with known attack patterns (signatures) to detect intrusions. It is also known as misuse detection. The advantage of this method is high detection accuracy for known attacks. The disadvantage is that it can only identify known attacks, and

zero-day attacks cannot be detected. [5]

- **Hybrid:** Combines anomaly and signature detection to obtain better detection performance.

2. **IDS Types based on Deployment Location**

- **Host based IDS (HIDS):** Monitors the events occurring within the host. Data sources to detect intrusions are process identifiers, system calls, registry accesses etc. [6]
- **Network based IDS (NIDS):** Monitors network traffic at selected points in a network. The data sources for detecting intrusions are traffic packets, network, transport, and application layer protocols. [6]
- **Distributed or hybrid IDS:** Monitors both host and network traffic to better identify intrusion activities. [6]

2.2 Deep Learning

Deep learning is a branch of machine learning that mimics the human brain, learning from examples. It works in an end-to-end fashion, in other words, it can extract features automatically. Accuracy that was not possible with previous methods can be obtained with deep learning. It can even exceed human-level performance in some tasks. Deep learning has a long history, but it was not popular in the past as today, since sufficient data and computation power did not exist in those days.

Biological nerve cells sum up the information taken from other cells that feed into it by dendrites. Then, the summed information is transferred through the axon and delivered to other cells by the synapses. Information is transmitted in the form of nerve impulses [7]. The shape of a biological neuron is shown in *Figure 2.2*.

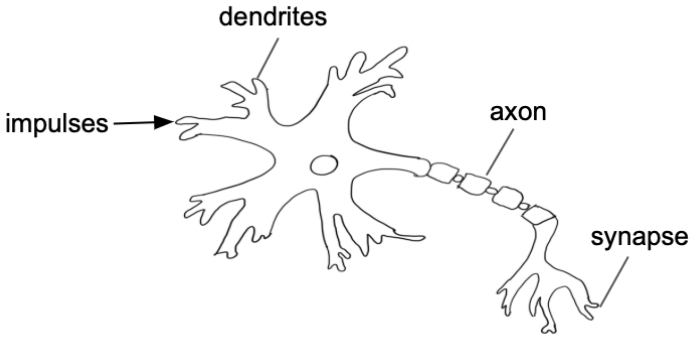


Figure 2.2: Biological neuron

The perceptron, or artificial neuron, which is a mathematical function, was introduced by Frank Rosenblatt in 1957 [8] inspired by the biological neuron. It creates a weighted sum by multiplying input values with weights and adding bias to them. Then, the activation function is applied to that sum to map the output between (0,1). The perceptron model is shown in *Figure 2.3*.

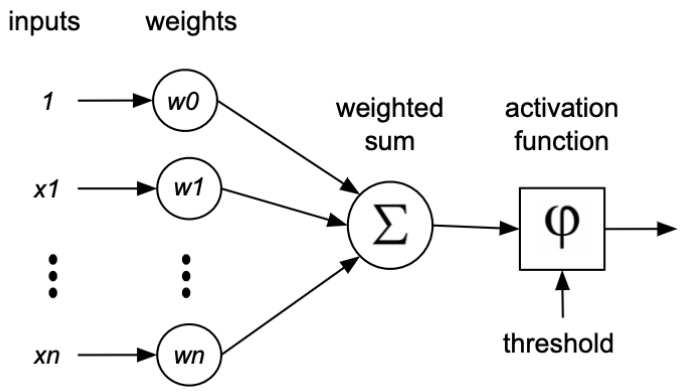


Figure 2.3: Perceptron (Artificial neuron)

Deep learning methods use neural network architectures which imitate the biological human brain. Multi-layer perceptron (MLP) or artificial neural network (ANN) was developed by Aleksei Grigoryevich Ivakhnenko and Valentin Grigorevich Lapa in 1965 [9]. Neural networks consists of at least 3 layers as shown in *Figure 2.4*: input layer, hidden layer and output layer. There is no limit to the count of hidden layers,

and depth is defined by the number of hidden layers in the neural network.

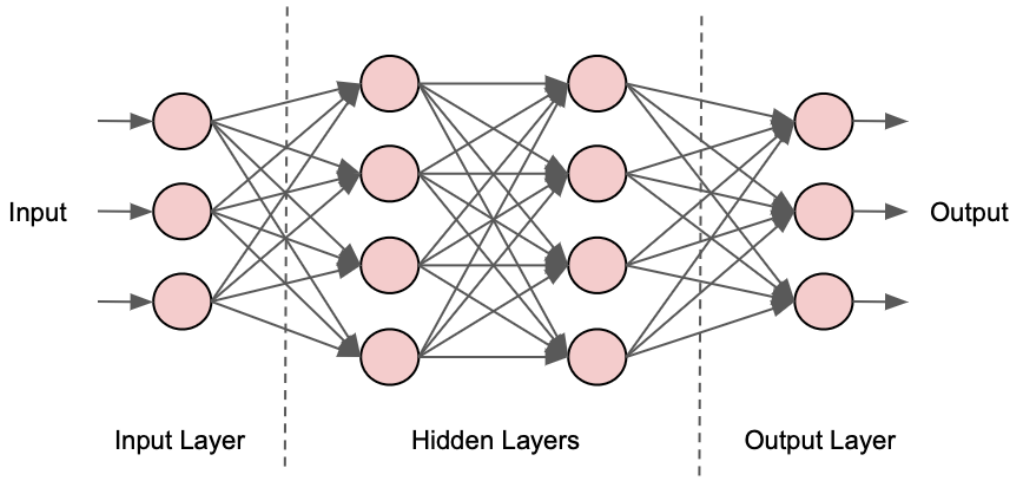


Figure 2.4: Neural network structure

The computation process from the input layer to the output layer is called forward propagation. There is another process that works in the backward direction, called backpropagation, introduced by Henry J. Kelley in 1960 [10]. It uses optimization algorithms like gradient-decent to calculate and minimize prediction errors. Weights and biases of the neurons in each layer of the network are adjusted with this process during training.

2.3 Active Learning

2.3.1 What is Active Learning?

For a wide variety of learning problems, obtaining a sufficient amount of labeled data is one of the most important challenges. Unlabeled data may be abundant or easily obtained, but labels for training are difficult, time consuming, or expensive to obtain in many learning problems. Active learning is a way to overcome the labeling bottleneck. The main motivation behind active learning is that if the learning algorithm is allowed to choose the data from which it learns, it can reach high accuracy

with fewer labeled training data instances. The active learner selects from unlabeled data instances as few as possible. Then, it asks queries in the form of unlabeled data instances to be labeled by an oracle (a human annotator). In this way, the cost of obtaining labeled data can be minimized [11]. *Figure 2.5* shows the active learning cycle.

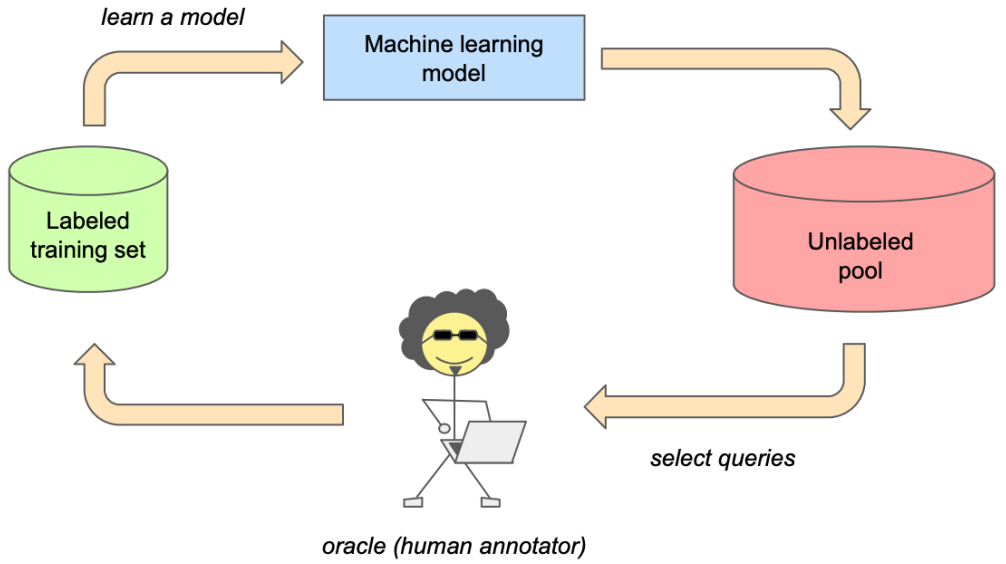


Figure 2.5: Active learning cycle

In fact, all data instances are not equally important for training. Some of them may be more representative and closer to the boundaries between classes. On the other hand, some data instances may be noisy or contain no useful information. Active learning can separate noisy and redundant data instances, and achieve even higher accuracy [12]. The study in [13] shows that active learning can provide higher accuracy.

There are two main scenarios for the active learning process [11]:

1. **Stream-based sampling:** Each unlabeled instance is drawn once at a time from the data source, and the active learner must decide whether to query or discard it.

2. **Pool-based sampling:** The large pool of unlabeled instances is available before

the active learning process, and queries are selectively drawn from that pool.

2.3.2 Query Strategies

How do we decide which instances will be chosen to label from unlabeled data for obtaining the most effective training dataset?

There is more than one answer to this question. In this section, the query strategies for the active learner will be explained [11].

1. **Uncertainty Sampling:** Probably the simplest and most commonly used query strategy is uncertainty sampling[14]. The instances that are most uncertain are queried by the active learner. This strategy is well suited for probabilistic learning models, but also applicable for non-probabilistic classifiers and regression problems. There are three different ways of calculating uncertainty.
    - **Least-confident strategy** queries instances whose prediction is least confident. The strategy is straightforward, but the weakness is that it only considers information about the most probable label.
    - **Margin strategy** queries instances according to the difference between the posterior of the first and second most likely label. It is a way of fixing the weakness of the least confident strategy.
    - **Entropy** is a more general uncertainty sampling strategy since it considers the probabilities of all possible labels. It is an information-theoretic measure. Shannon entropy [15] is the most popular one for active learning.
2. **Query by Committee (QBC):** The approach maintains a committee of models that are all trained on the current labeled dataset. Each committee member predicts the labels of unlabeled data instances. The instance which has most disagreement on is considered the most informative data, and the active learner queries those data instances to label [16].
3. **Expected Model Change:** Expected model change is a decision-theoretic approach. It queries the instance has the highest potential to change the current

model most if we knew its label. Expected gradient length (EGL) [17] is an example for this approach. The "change" on the model can be measured by the gradient length for the models trained using gradient-based optimization.

4. **Expected Error Reduction:** Expected Error Reduction is also a decision-theoretic approach. It queries the instance that is the most likely to reduce the generalization error of the model. A data instance from an unlabeled data pool is added to the training set temporarily, and the model is trained with that dataset. Then, the expected future error for that instance is estimated on the remaining unlabeled instances with the new model. The approach is less prone to querying outliers since it considers the entire input space rather than individual instances.

5. **Variance Reduction:** Minimizing the expectation of a loss function cannot be done in closed form in general. The generalization error can be reduced by minimizing the output variance. This is a more efficient way to reduce expected errors, since it sometimes has a closed-form solution. The approach is less prone to querying outliers since it considers the entire input space rather than individual instances.

6. **Density Weighted Methods:** The idea behind density-weighted methods is that informative instances are not only those which are uncertain but also those that are "representative" of the underlying distribution. In this approach, the informativeness is measured with a weighted version of the base query method. Weights are calculated according to the average similarity of the data instance to all other instances in the input distribution. The method was formulated by Settles and Craven [18].

2.4 Incremental Learning

Humans have a lifelong learning ability to adapt to changing environments. They accumulate information from their environment, fine-tune it, and make inferences for new tasks by using previous knowledge throughout their lifespan. This ability is not only critical for us, it is also crucial for computational learning systems and

autonomous agents. Nevertheless, incremental learning for machine learning and neural network models is still an unresolved issue. They have achieved human-level performance on individual tasks, but they are static models and incapable of adapting their behavior over time. Incremental learning from non-static data distributions generally causes catastrophic forgetting [19]. New information overwrites previously learned knowledge partially or completely in the shared representational resources of the model when new data instances are significantly different from previously learned data instances. This is called catastrophic forgetting [20]. Therefore, they require retraining from scratch each time new data becomes available for adaptation to changes in the data distribution. This is the main deficiency of classical deep neural network models. The incremental learning process is shown in *Figure 2.6*.

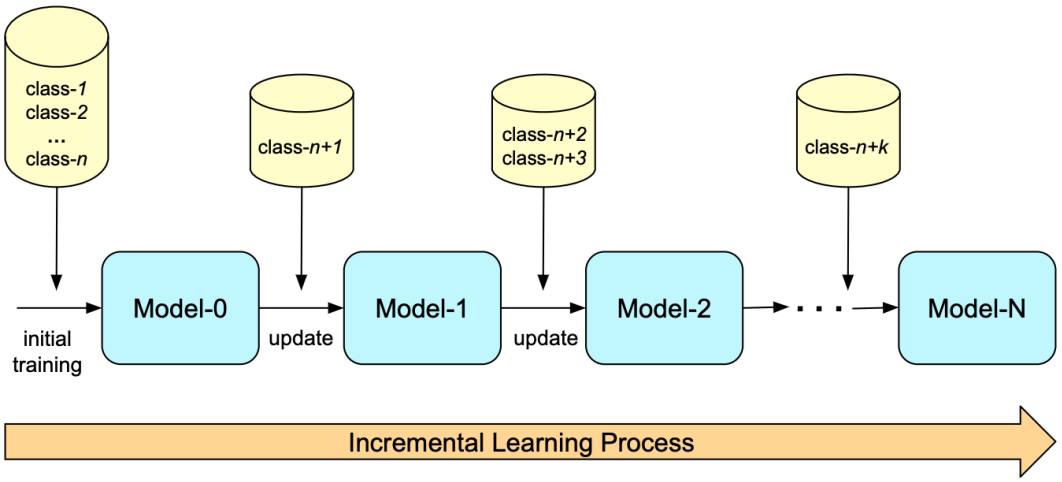


Figure 2.6: Incremental learning process

There are three basic incremental learning approaches based on how task-specific information is stored and used [21]:

1. **Replay methods:** Previous data instances are stored in raw format and replayed while learning a new task to overcome forgetting. They are either reused as training data or to constrain optimization of the new data instances' loss. The most popular algorithms for this approach are class incremental learner, iCaRL [22] and Gradient Episodic Memory, GEM [23].

2. **Regularization based methods:** Instead of storing previous data instances, this approach introduces an additional regularization term in the loss function to consolidate previous knowledge when learning new data. The most important algorithm in this family is Elastic weight consolidation, EWC [24].

3. **Parameter isolation methods:** This approach changes architectural properties by increasing the number of neurons or network layers to prevent any possible forgetting. When there is no constraint on architecture size, a neural network can allocate a new branch for each new task and freeze the previous task parameters. The most popular algorithm in this family is PackNet [25].

CHAPTER 3

RELATED WORK

3.1 IDS with Deep Learning

The challenges of improving performance in terms of detection accuracy, false alarm rate, and dealing with unseen attacks still exist for current IDSs. Using deep learning methods to increase the performance of IDS has been a very popular research area in recent years. Various kinds of methods have been tried, and the results show that deep learning methods exhibit remarkable performance compared to traditional machine learning methods. Some of the popular survey articles about the use of deep learning for IDS are reviewed in this section.

The survey [26] proposes a taxonomy for IDS based on machine learning and deep learning. They summarize representative studies between 2015 and 2019 and compare them empirically. Machine learning-based IDSs have substantial detection and generalization performance when a sufficient amount of training data is available. However, deep learning-based IDSs can give better results. In the problem of dealing with big data, deep learning methods are better than traditional machine learning. Moreover, deep learning methods can extract feature vectors from raw data. Therefore, they are easy to use since there is no need for manual operations during the process, and they just get raw inputs and give the prediction results. Because of the reasons mentioned before, deep learning methods have great potential and popularity in the field of IDS.

In the paper [27], Gamage et al. categorize deep learning models for IDS and summarize them. The taxonomy is shown in *Figure 3.1*

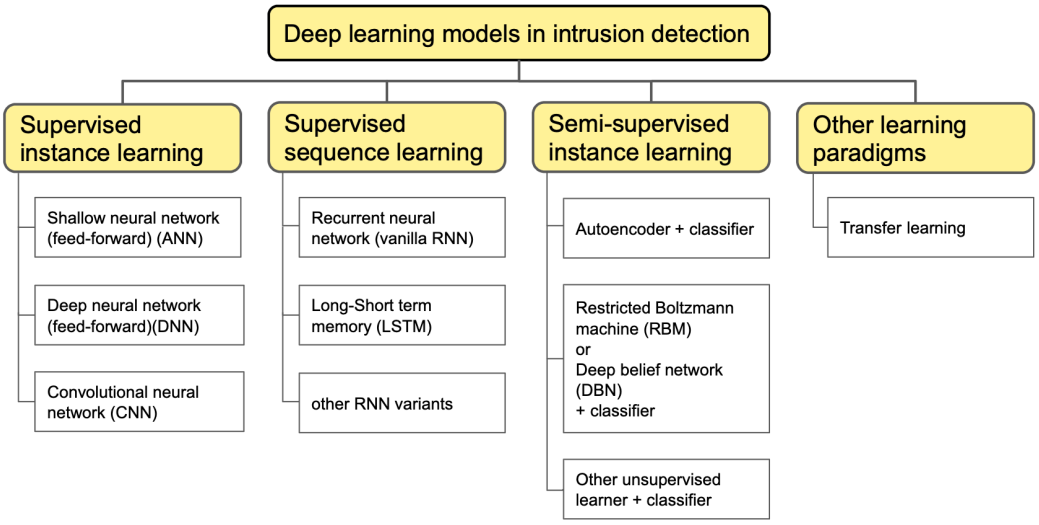


Figure 3.1: Categorization of deep learning methods in IDS

They worked on four main deep learning methods: feed-forward neural network, autoencoder, LSTM, and deep belief network for the attack classification problem. Additionally, a random forest which is a classical machine learning method commonly used for intrusion detection is also used for comparison. Two old datasets (KDD-99, NSL-KDD) and two new datasets (CICIDS2017, CICIDS2018) are selected for training and evaluation of the models. The results of the experiments show that random forests give good results on the new datasets. Their implementation is easy and training time is short. On the other hand, their prediction time is high for a real-time system. ANNs are faster than random forests by a factor of 5-10 times.

The most important finding in this study is that LSTM networks and semi-supervised models (AE + ANN, DBN + ANN) do not give better results than ANNs. The implementation and the experiments are shared with the research community for further use and analysis. The released implementation of the paper is taken as basis in this thesis.

3.2 IDS with Active Learning

Collecting huge amounts of unlabeled data for IDS is easy, but high-quality labeled data cannot be obtained easily. The labeling process must be made by security experts and requires a lot of time. Because of that, the labeling is expensive. A substantial amount of labeled data is needed to construct an IDS that has sufficient performance. However, available labeled data is very limited. This is one of the most important bottlenecks for developing high-performance systems. The most important motivation behind using active learning for IDS is that the same performance may be achieved with a smaller labeled dataset. Moreover, there is a possibility of achieving even much better performance by using small amounts of representative, noiseless, high-quality labeled data instead of huge amounts of labeled data in different quality. The researchers applied different machine learning algorithms with active learning to construct IDS and compared the performance of the system with different amounts of data. Although a limited amount of work exists combining active learning and IDS, the experiments from the studies show that results are promising to achieve this goal.

The first study applying active learning to IDS is [28]. They propose an IDS by using SVM as a classifier with active learning. The method is evaluated on the KDD'99 dataset, and the results show that active learning can reduce the number of labeled data instances significantly although it conserves the performance of a traditional classifier that needs much more labeled data. Moreover, the required labeled data instances can be reduced to even 20% of the total data instance count for some cases.

[29] proposes an IDS by using the TCM-KNN (Transductive Confidence Machines for K-Nearest Neighbors) algorithm with active learning. They use the KDD 99 dataset for their experiments. They observed that active learning is a better option than random sampling. The same accuracy can be achieved using 40 data instances selected by active learning and 2000 data instances selected by random selection.

The proposed method in [30] consists of a classifier module based on the Metacost algorithm which is a cost-sensitive learning method and an active learning module using the largest misclassification cost as a sampling criterion. The results of the experiments on the KDD'99 dataset show that the active cost-sensitive sampling method

reduces the required labeled instance count to 16% of the total data when compared with the standard random sampling method.

[31] propose an anomaly-based IDS by using semi-supervised learning and co-training methods. The Naive Bayes classifier is used for supervised learning. They evaluated their method on the KDD'99 dataset. The experiments show that active labeling can significantly decrease the error rate by using 6.6% of labeled data.

[32] propose an anomaly-based IDS using neural networks as a classifier with active learning. They evaluated their method on the KDD'99 dataset. They use a small portion of the dataset as an initial training dataset and iteratively additional data instances selected by active learning were added to the training dataset. They reported that the active data instance selection yielded better performance than the random selection of instances.

[33] propose an adaptive false alarm filter for NIDS by using active learning. The KNN (k-nearest neighbor) algorithm is used as a classifier in this work. The experimental results show that the method can reach higher performance than a classical machine learning method, and reduce the required amount of labeled data instances to 50%.

[34] propose an IDS using a hybrid semi-supervised machine learning technique. The method combines Active learning Support Vector Machine (ASVM) and Fuzzy C-Means (FCM) clustering. NSL-KDD dataset is used for evaluation. Results show that active learning has the potential to improve IDSs.

[35] is the first study applying active learning to IoT IDS. Their system detects outliers first in an unsupervised way. Then iteratively, the system selects from unlabeled data and labels them by an oracle until the system reaches expected performance in terms of precision and recall. They evaluated their method using the KDD'99 dataset. They observed that an active learning-based method converges to the target performance much quicker than the random one. In addition, the same performance is achieved with 30% of labeled data instances by using active learning methods. This is a significant improvement in the efficiency of the labeling process.

[36] proposes an active transfer learning based IDS, called ACTrAdaBoost. Experiments on the DARPA1998, the KDD 99, and the ISCX2012 datasets show that the method gives better performance when active learning is used.

The nature of the intrusion detection problem requires real-time or near real-time response, and quick adaptation to new attacks. Therefore, reducing the training time is essential for a real-time system if it needs to be retained continuously [32]. A long training time is needed to train with a huge amount of training data. In fact, there is no need for all instances in a large training set since it also contains redundant, noisy data instances. Therefore, using active learning is a good way of shrinking the training set by choosing the most informative instances [37]. Some works are focused on using active learning for updating the classifier.

[38] proposed an IDS consisting of two modules: a random forest classifier and an active learner trainer. Classification predictions are analyzed and some of the data instances are selected to label by an oracle. After new data instances are labeled, the classifier is retrained with a new training dataset. Those processes are monitored by the trainer module. K-means clustering algorithm is used to select which sample will be labeled. The oracle is simulated with another random forest classifier. KDD 99 dataset is used for evaluation in this study. The dataset is split into 49 batches representing days. The learner observes the data instances that just belong to the current day. Some of the data instances of the day are chosen for labeling, and the classifier is retrained with a training set containing those new data instances. Experiment results show that normal class data can be detected with 90% accuracy when only 0.13% of data instances selected by the learner are used for training. However, the detection accuracy of other attack classes is lower than the normal class since there are fewer data instances belonging to those classes. Active learning provides a more efficient manual labeling process. Moreover, the method enables the adaptation of classifiers for changing data.

[39] builds on their previous work [38]. The problems related to event level IDS and the large amounts of intrusion alerts are analyzed by using active learning and cyber situation awareness. The authors conducted two experiments on the UNSW-NB15 dataset. In the first experiment, a random forest classifier is trained with the data

instances selected by active learning, and active learning methods are evaluated. The second experiment is about cyber situation awareness. The detection results of the first experiment are aggregated and the probability of a computer system being part of an attack is calculated. The results of those experiments show that the probability of a computer system being involved in an attack could be calculated cumulatively with a high degree of accuracy after just a few active learning cycles.

[40] proposed a hybrid IDS which is adaptive to unseen attacks. The method consists of two modules: a supervised module using Deep Neural Network (DNN) and an unsupervised module using K-Nearest Neighbors (KNN). Both supervised and unsupervised modules perform binary classification (attack vs. normal). They used the least-confident uncertainty method as an active learning strategy for teaching the supervised module. The Supervised module, DNN is retrained incrementally. They used the CICIDS2017 dataset for their experiments. The dataset is divided into training and a test set, using a 70 - 30 $\%$ split, respectively. The training set was split into sliding windows that contained 5000 samples. The first window is completely labeled and used for pre-training of the model. The others that include new attack types are used for incremental learning. They analyze the effect of queried sample size in each window on system performance and compare them with the baseline trained by using the full training data. When queried sample size increased, in other words, more labeled training data was provided, they got better DNN performance. In addition, they observed that labeling only 13% of the full training data selected by active learning is enough for the performance of the baseline which is trained with the full training set.

In [38] and [39], the classifier is improved with active learning progressively, but a new class of attack is not added. Furthermore, they just analyze the effect of the selected sample size for labeling and minimum confidence threshold. Other parameters affecting the performance are not researched. For the first time, active learning is used to achieve adaptive IDS in the study [40]. Their system permorms binary classification. Therefore, the system learns to distinguish new kinds of attacks from normal data, but it cannot learn to classify them as additional types of attacks since it is not a multi-class classifier. Moreover, they just analyze the effect of the selected instance size from the pool and do not analyze the effects of all parameters that the system depends on. In addition, they used just the least-confident uncertainty method

and did not examine other active learning methods. Because of that, their work is not comprehensive.

Another advantage of using active learning is that it is resistant to poisoning attacks. IDSs based on machine learning methods have the potential to be an objective of an attack. Attackers' activities may not be detected by IDS after those kinds of attacks [38]. The main idea behind poisoning attacks is that the attacker injects fake unlabeled data which makes attacks classified as benign by the model. New training data must be involved for the adaptation of IDS to changing environments. Since a sufficient amount of labeled instances for new attacks is not available generally, semi-supervised learning methods are an alternative to enhance attack classification. However, they introduce new security vulnerabilities with unlabeled data. Long et all show that the semi-supervised classifier could be misled by the attacker. Moreover, they developed a defense mechanism using active learning for those kinds of attacks. They observe that the detection performance of the classifier based on semi-supervised learning can be decreased with poisoning attacks. Experiment results show that their method achieves better performance than the classifier based on semi-supervised learning when a poisoning attack is occurs.[41]

There is just one survey article that summarizes studies about IDS with active learning [37]. Authors conclude that most of the work combining active learning with IDS is based on SVM, ANN, or KNN classifiers. The most popular active learning strategy is uncertainty sampling. The publications combining active learning with intrusion detection are very limited. Also, this is an old survey study, and a survey comparing results of current works does not exist.

The work in [42] compares the effect of active learning strategies on intrusion detection systems. Three active learning strategies are selected: uncertainty sampling, expected model change, and Query By Committee (QBC). They used the Neural Network (NN) for binary (normal and attack) classification. They evaluated the classifier on the KDD CUP 1999 dataset by training with the data selected by different active learning methods. hey observed that the expected model change strategy gives the best performance over other strategies. Uncertainty sampling is the second one, and QBC is in the last place. Results give some intuition but they are not giving a direct

opinion about performance comparison on the modern network attacks since an old dataset is used for evaluation. In addition, there are still unexamined active learning methods for intrusion detection.

3.3 IDS with Incremental Learning

Deep learning-based IDSs achieve high detection performance. However, they are usually trained once before deployment and have the ability to classify only known classes at train time. When new attack types arise, those systems fail to classify them. The classification model must be retrained from scratch to be able to learn additional attack types. This is not an efficient method since IDS needs quick adaptation to changes. Using incremental learning for updating IDSs efficiently is a new research area. As far as we know, all of the work about incremental IDS is from the last two years. In this section, those studies are reviewed.

In [43], Zhang et al. developed an IDS which is scalable and adaptive to unknown attacks. The system consists of three components. The first one is the open set classification network which is used to detect unknown attacks. The classification network is based on CNN and the nearest class mean classifier. The second component is the semantic embedding clustering which is used to find out obscure unknown attacks which couldn't be detected by the classifier. The last one is the incremental nearest cluster centroid module. This module updates the classifier with new attack types. They evaluated the system on the KDD 99 and the CICIDS2017 datasets. The system works better than other state-of-the-art methods in terms of the detection of multiple unknown attacks.

In [44], Martina et al. propose an IDS, called Soft-Forgetting Self-Organizing Incremental Neural Network (SF-SOINN). The method has incremental learning ability. In addition, it has a short classification time and high performance. Some of the neurons in the model are removed based on utility measures. NSL-KDD and CIC-IDS-2017 datasets are used for validation of the method.

In [45], Amalapuram et al. apply two popular incremental learning methods Elastic Weight Consolidation (EWC) and Gradient Episodic Memory (GEM) to IDS. A sim-

ple Multi-Layer Perceptron (MLP) and simple Convolutional Neural Network (CNN) are used as a classifier. The performances of the methods across different task orders are evaluated on CICIDS-2017, CICIDS-2018, and KDD'99 datasets. The results show that the proposed architecture is one of the most scalable solutions.

In [46], Lin et al. propose a two-stage IDS using Deep Neural Network (DNN) and Incremental Learning (IL). The model was trained using CAN data in the offline stage. Afterward, the model is updated with new data by using incremental learning methods in the online stage. The method has high performance according to experimental results.

CHAPTER 4

METHODOLOGY

4.1 Architecture and General Properties of Proposed IDS

The architecture of the proposed IDS consists of four modules: the classification module, the incremental learner module, the active learner module, and the simulated oracle module. In this section, interactions of the modules and the general functioning of the system will be explained. *Figure 4.1* shows the general structure of the architecture.The details of the modules will be clarified in the next sections.

The classifier gets the feature set as input and gives the label the highest probability as the output according to the prediction of the neural network model. If the resulting label has less probability than the minimum confidence threshold, then the input feature set is added to the unlabeled data pool. Regardless of the confidence of the label, the result is returned. Note that the neural network model is pre-trained with a small train dataset as an initial step before starting the system.

Whenever the unlabeled data pool size exceeds the pool size threshold, the incremental learning module is activated. The incremental learner module gives the unlabeled data pool to the active learner module and requests the selected data instances with their labels from it.

The active learner module calculates a score for each data instance in the unlabeled data pool according to the chosen query method. Then, some of the instances are selected using the scores according to the chosen selection criteria. The module communicates with the simulated oracle and asks for the labels of the selected unlabeled data instances.

Afterward, the selected data instances given by the active learner module are split into two parts by the incremental learner module: known class data and new class data. Known class data is directly added to the training dataset, but new class data is added to the rare class dataset first. Then, instance count for each class in the rare class dataset is checked. If there exist data instances as many as the min count for the new class threshold, the data instances are transferred from the rare class dataset to the train dataset. New classes are waited before use until the specified data instance count is reached.

If a new class is added to the training dataset, the output layer of the neural network model must be extended for those classes. Therefore, new neurons are added to the output layer as many as the number of new classes. However, the weights of the neurons in the hidden layers are preserved in order not to forget the learned knowledge so far.

The neural network model is retained with the updated training dataset by the incremental learner module. After retaining is completed, the unlabeled data pool is made empty to be ready for the next cycle. After each retaining cycle, the model is improved for known classes and also learned new classes. The cycle continues as long as the system is running, but the period is expected to increase since the system will be more stable after some point.

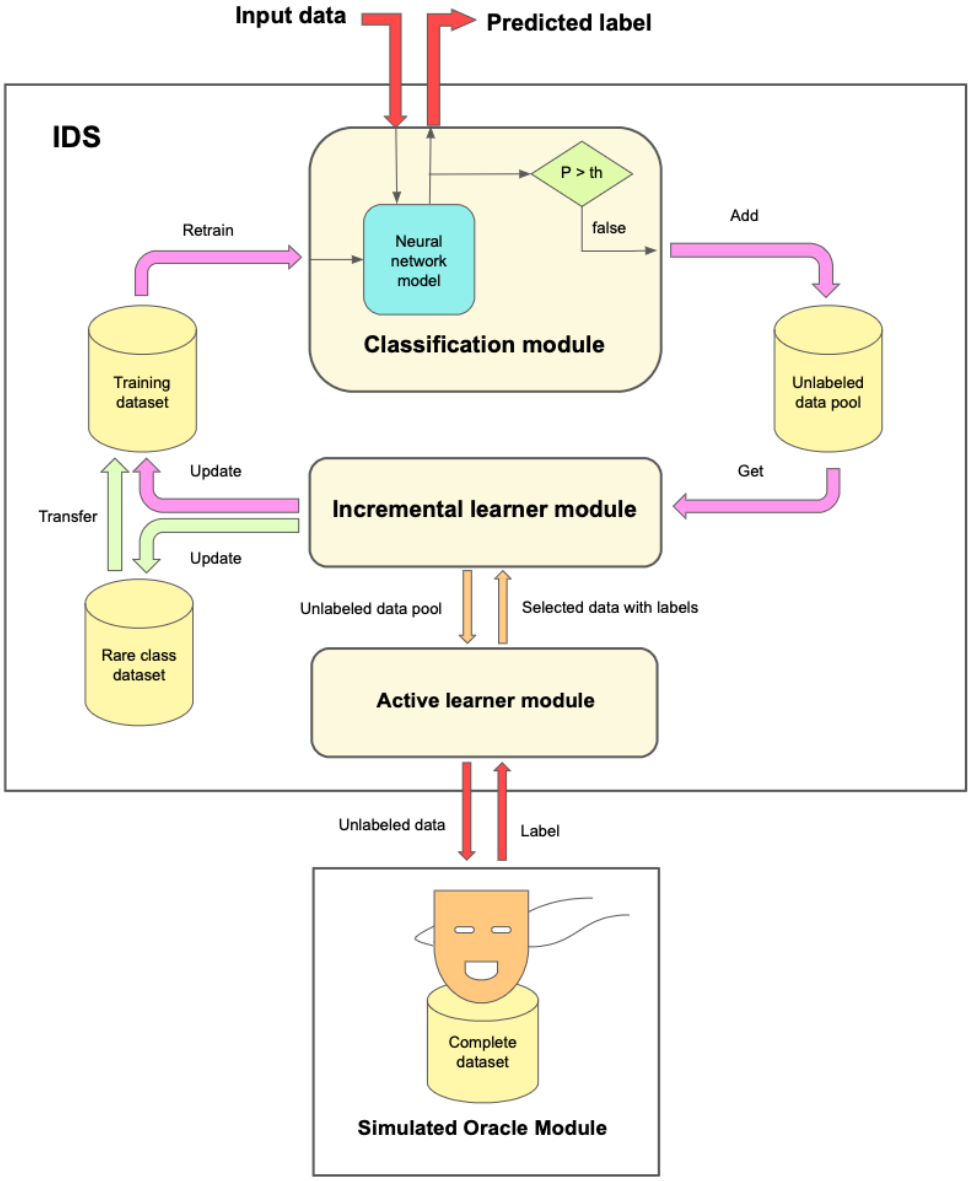


Figure 4.1: The architecture of the proposed IDS

4.2 Classification Module

Gamage et al. found that any of the commonly used deep learning methods for IDS do not outperform the ANN [27]. Based on the findings of the study, ANN is chosen for the classification method. The architecture is shaped according to the experiment results of the work. In addition, the shared implementation of the study is used as a basis.

The number of neurons in the input layer is adjusted according to the feature count in the selected dataset. The experiments of the paper show that the best performance is obtained from ANN with a single hidden layer having 64 neurons and four hidden layers having 256-128-64-32 neurons respectively. Both of the architectures give almost the same performance. Therefore, a single hidden layer with 64 neurons is used in this work. RELU is used as an activation function in the hidden layer. The number of neurons in the output layer is adjusted according to the class count in the training dataset. The output layer activation function is softmax.

Dropout and batch normalization are applied to overcome overfitting. The dropout rate is 0.20 since it gives the best performance in this setting according to the study. The cross-entropy loss function is used with the Adam optimizer. Class weighting is applied to deal with the imbalanced nature of intrusion detection data. Class weights are calculated according to the instance count of class in training data before every retraining process.

The classification module is the main module in the system. It predicts labels of the input data and also collects the data for which there is not enough confidence in the predicted labels.

4.3 Incremental Learner Module

The responsibility of the incremental learner module is to update the neural network model for adapting to the changing environment. It picks up new data instances with their labels by communicating with the active learning module, manages the new data, and retrains the model.

4.4 Active Learning Module

The active learning module is configurable in terms of query and selection strategy. The active learning module supports three types of uncertainty query methods: least-confident, margin, and entropy. A score is calculated for each unlabeled data instance by using the query method.

- **Least confident:** Uncertainty measure for this strategy is calculated according to the probability of the label has the highest probability among possible labels. The formula is

$$\mathbf{S}_x = 1 - \mathbf{P}(\hat{\mathbf{y}}|\mathbf{x}) \tag{4.1}$$

  where $\hat{y}$ is the most likely label for data instance x.

- **Margin:** Difference between highest probability and second highest probability among possible predicted labels gives the uncertainty measure in this strategy. The formula is

$$\mathbf{S}_x = \mathbf{P}(\hat{\mathbf{y}}_\mathbf{1}|\mathbf{x}) - \mathbf{P}(\hat{\mathbf{y}}_\mathbf{2}|\mathbf{x}) \tag{4.2}$$

  where $\hat{\mathbf{y}}_\mathbf{1}$ is the most likely label and $\hat{\mathbf{y}}_\mathbf{2}$ is the second most likely label for data instance x.

- **Entropy:** Entropy considers the probability of all possible labels when calculating uncertainty measures. Shannon entropy [15] is preferred in this work since it is the most popular entropy calculation method. The formula is

$$\mathbf{S}_x = -\sum_i \mathbf{P}(\mathbf{y_i}|\mathbf{x})\, log_2\, P(\mathbf{y_i}|\mathbf{x}) \tag{4.3}$$

  where $\mathbf{y_i}$ ranges over possible labels for data instance x.

Selection methods have two kinds: maximum *n* instance and minimum *n* instance. It means the first data instances that have a maximum or minimum score are selected.

4.5 Simulated Oracle Module

An identifier number is given to each data instance in the dataset. All of the data labels in the dataset is given to the simulated oracle module with the identifier numbers. When asked label of any data instance, simulated oracle searches the label by using the identifier number and gives the label. Note that data instance is given as input with the identifier number.

A simulated oracle module is used for experimental purposes since working with a security expert was not possible within the scope of this thesis. In real use cases, instead of the simulated oracle, real security experts will label the data.

CHAPTER 5

EXPERIMENTS AND DISCUSSION

5.1 Dataset

The CICIDS-2017 intrusion detection dataset [47] is used for evaluation. The dataset was captured in a 5 day period from 3 to 7 July 2017. There are 80 flow-based traffic features, and 2 of them are text-based. The dataset is fully labeled and contains 15 different labels from seven common families of attacks: Botnet, Brute Force Attack, DoS Attack, Heartbleed Attack, Infiltration Attack, DDoS Attack, and Web Attack. Benign background traffic is generated using more than 20 users' behavior based on the FTP, HTTPS, HTTP, SSH, and email protocols. The generated network traffic represents a modern network traffic scenario.

Data pre-processing: Just numeric features were used in this work. Thus, 78 features exist in the pre-processed dataset. In addition, the data belonging to 3 of the labels were removed from the dataset since their instance count is too few. The paper [27] shows that the complete dataset and 10% of it give the same performance when the smaller dataset is prepared by preserving the original class proportions. Therefore, the dataset was reduced to 10 % by using the same approach. Then, 20 % of the small dataset was separated for testing. Initially, some portion of the remaining part was separated for pre-training of the model. The proportion was configured for different values. The pre-training data includes only 3 classes: BENIGN, PortScan, and DDos, and other classes were removed. The remaining data after those processes was given as input to the IDS sequentially in 100 data-instance batches. The reason for giving input as a batch is to speed up the experiments since the calculations take much more time when it is given to the neural network one by one. Moreover, it is a more realistic

scenario, because more than one data flow exists at the same time in the network and they are checked by IDS.

5.2 Experimental Setup

The proposed IDS was implemented in Python language, and ANN was built by using Tensorflow and Keras. The ANN trained with the complete dataset was used as a baseline. The baseline was trained with a batch size of 256 and epoch size of 150 at once.

Experiments were designed according to the parameters that can affect performance:

- **initial data split ratio:** the percentage of the dataset used for pre-training
- **batch size:** training batch size of the neural network
- **epochs:** training epoch count of the neural network
- **query strategy:** active learning strategy used for calculating uncertainty measure
- **selection strategy:** maximum or minimum score
- **selection count:** number of data instances to select from the unlabeled data pool
- **pool size threshold:** unlabeled data pool size for activating the active incremental learning process
- **new class min count threshold:** minimum required number of data instances to include a new class in the training dataset

Default configurations containing those parameters are prepared by using initial tested values. Then, test configurations generated by changing the values of the parameter will be analyzed while the others are constant. Default configuration values are listed below:

- initial data split ratio = 0.05
- batch size = 256
- epochs = 10
- query strategy = least-confident
- selection strategy = max-n
- selection count = 100
- pool size threshold = 500
- new class min count threshold = 1

The experiments were performed in the Turkish National e-Science e-Infrastructure (TRUBA) environment. High-performance computing and data storage are provided for all research institutions and researchers in our country by TÜBİTAK ULAKBİM High Performance and Grid Computing Center.

5.3 Evaluation Measure

Performance measures and their formulas are listed below:

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $F1\ score = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$

True positive(TP) represents number of outputs which are predicted as the positive class correctly. Similarly, true negative(TN) represents number of outputs which are predicted as the negative class correctly.

False positive(FP) represents number of outputs which are predicted as the positive class incorrectly. Similarly, false negative(FN) represents number of outputs which are predicted as the negative class incorrectly.

In addition to the performance measures, some data count information is recorded for further analysis:

- data counts for each class in input
- data counts for each class in unlabeled data pool
- data counts for each class in selected data
- data counts for each class in training data

By comparing input and unlabeled data pool content, we can analyze classes the system is not confident about. A comparison of the unlabeled data pool and selected data content gives the selection behavior of the system. Examining training data contents provides a better understanding of the system performance.

Note that performance measures and data count information are recorded for each learning process.

5.4 Results and Discussion

We conducted experiments with 3 methods. There are 6 parameters that affect their performances. For each method, the following issues will be explained for the system in the default configuration:

1. Evaluation graph for the default configuration
2. General performance measure change
3. Change of performance measure for each class
4. Data analysis for adding data instances to the unlabeled data pool
5. Data selection behavior analysis
6. Train data updates behavior

Then, the comparison graphs for different values of the parameters will be explained for the 6 parameters and each method. Lastly, the 3 methods will be compared in terms of performance and data usage in the general discussion section.

5.4.1 Least-confident Method Results

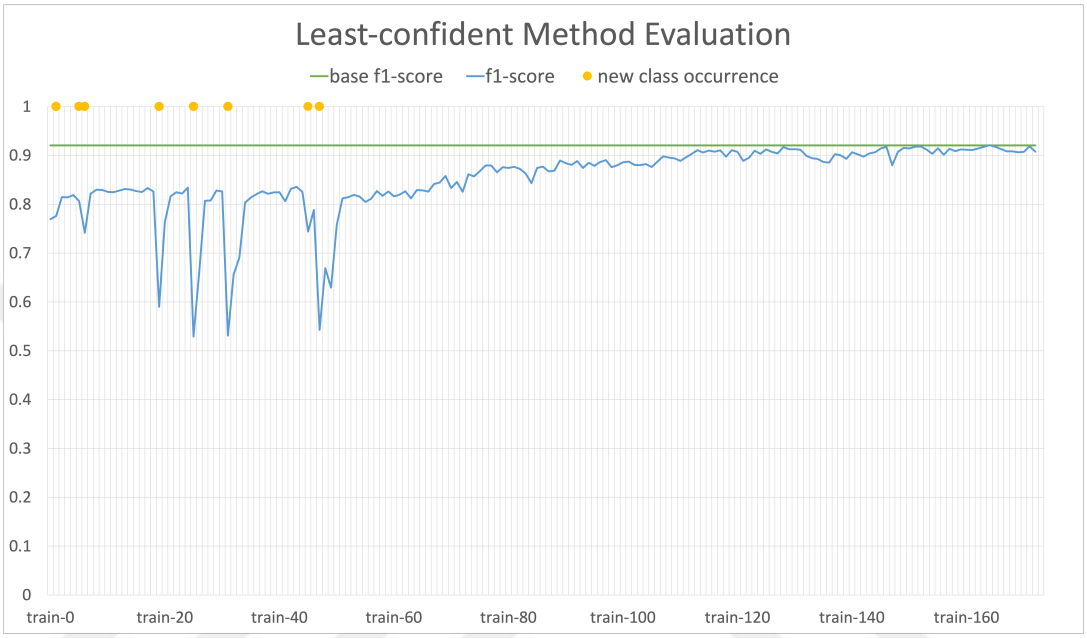


Figure 5.1: Evaluation graph of least-confident method

The evaluation graph of the least-confident method is shown in *Figure 5.1*. In this graph, we can observe that the change in the f1-score of the IDS is based on the least-confident active learning strategy. "train-0" means pre-training cycle, and retraining cycles are named as train-x. In each retraining cycle, new data instances selected by the active learner are added to the training dataset, and the classifier is updated accordingly. Yellow dots show the places where a new type of attack is added to the dataset. A new type of class was added 8 times. As shown in the graph, some amount of performance decrease occurs when a new class is added to the dataset since the new class introduces additional confusion between classes. However, after a few learning cycles, new data instances belonging to the new class are added to the training dataset, and the performance is increased progressively. The green line shows the f1-score of the baseline which is the classifier trained with whole train data instances at once.

We can see that the performance of the classifier converges to the performance of the baseline in approximately train-110.

Table 5.1: Least-confident method's general performance table

name	accuracy	accuracy balanced	f1-score weighted	f1-score macro	precision weighted	precision macro	recall weighted	recall macro
train-0	0.833	0.149	0.770	0.125	0.717	0.109	0.833	0.149
train-1	0.835	0.160	0.776	0.134	0.745	0.144	0.835	0.160
train-5	0.836	0.263	0.807	0.227	0.788	0.213	0.836	0.263
train-6	0.712	0.261	0.742	0.197	0.797	0.174	0.712	0.261
train-19	0.517	0.321	0.590	0.157	0.829	0.174	0.517	0.321
train-25	0.439	0.370	0.529	0.179	0.797	0.183	0.439	0.370
train-31	0.445	0.279	0.531	0.163	0.794	0.160	0.445	0.279
train-45	0.716	0.386	0.744	0.201	0.802	0.171	0.716	0.386
train-47	0.458	0.436	0.543	0.190	0.836	0.193	0.458	0.436
train-172	**0.894**	**0.723**	**0.908**	**0.541**	**0.927**	**0.530**	**0.894**	**0.723**
baseline	**0.913**	**0.620**	**0.920**	**0.527**	**0.934**	**0.560**	**0.913**	**0.620**

Change in all performance measures for the classifier based on least-confident active learning strategy is listed in *Table 5.1*. There are four performance measures: accuracy, f1-score, precision, and recall. Performance measures have two types: average and weighted average. An average measurement means the calculating mean of the measurements for each class. On the other hand, a weighted average measurement means the calculating mean of the measurements for each class weighted with their instance count in the test dataset. Weighted average shows the real use case performance since the performance of the most common classes affects the total performance more. However, the direct average represents the effect of the learning of a new class more clearly. "train-0" means pre-training cycle, and other train cycles are where a new type of attack is added to the dataset. In addition, the performances at the last training cycle and baseline are added to the end of the table. Values of all performance measures are increasing progressively and converging to the baseline as shown in the table. Moreover, the classifier reaches higher values for some of the measures in the direct average type although weighted versions are almost the same. It means the classifier learned some of the classes better than the baseline.

Table 5.2: Least-confident method's f1-score table for each class

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
train-0	0.926	0.000	0.569	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
train-1	0.927	0.000	0.607	0.000	0.000	0.000	0.000	0.000	0.073	0.000	0.000	0.000
train-5	0.924	0.000	0.555	0.000	0.000	0.563	0.000	0.000	0.686	0.000	0.000	0.000
train-6	0.835	0.000	0.603	0.000	0.089	0.198	0.000	0.000	0.639	0.000	0.000	0.000
train-19	0.652	0.039	0.535	0.000	0.307	0.057	0.000	0.000	0.301	0.000	0.000	0.000
train-25	0.564	0.004	0.553	0.000	0.144	0.123	0.052	0.019	0.692	0.000	0.000	0.000
train-31	0.579	0.001	0.521	0.067	0.064	0.000	0.035	0.035	0.651	0.000	0.000	0.000
train-45	0.853	0.065	0.374	0.046	0.000	0.132	0.000	0.000	0.729	0.213	0.000	0.000
train-47	0.583	0.013	0.531	0.192	0.170	0.039	0.031	0.047	0.628	0.005	0.041	0.000
train-172	**0.936**	**0.078**	**0.904**	**0.534**	**0.708**	**0.534**	**0.415**	**0.821**	**0.887**	**0.622**	**0.051**	**0.000**
baseline	**0.947**	**0.006**	**0.914**	**0.907**	**0.879**	**0.437**	**0.183**	**0.656**	**0.695**	**0.620**	**0.078**	**0.000**

Changes in f1-scores for each class are listed in *Table 5.2*. "train-0" means pre-training cycle, and other train cycles are where a new type of attack is added to the dataset. In addition, the performances at the last training cycle and baseline are added to the end of the table. We can see that new classes were learned incrementally and almost all classes were learned at the end. For some of the classes, the classifier performs better than the baseline.

Table 5.3: Least-confident method's input data and unlabeled data pool table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
input_data-1	556	0	23	3	61	1	0	3	52	1	0	0	700
unlabeled_data_pool-1	447	0	12	2	22	0	0	3	50	1	0	0	537
input_data-5	809	4	49	5	74	2	2	2	50	2	1	0	1000
unlabeled_data_pool-5	449	4	8	0	10	1	1	1	25	2	0	0	501
input_data-6	572	1	31	2	53	1	0	2	33	5	0	0	700
unlabeled_data_pool-6	522	1	12	2	16	0	0	2	21	5	0	0	581
input_data-19	1187	2	82	2	120	4	1	7	89	3	3	0	1500
unlabeled_data_pool-19	422	2	15	0	13	1	0	3	45	3	1	0	505
input_data-25	826	1	45	0	70	4	3	0	45	6	0	0	1000
unlabeled_data_pool-25	437	1	17	0	17	1	2	0	22	6	0	0	503
input_data-31	556	1	30	5	58	1	0	3	44	2	0	0	700
unlabeled_data_pool-31	460	1	11	3	14	0	0	3	23	2	0	0	517
input_data-45	970	0	56	5	96	2	1	3	64	3	0	0	1200
unlabeled_data_pool-45	494	0	7	3	10	1	1	3	28	3	0	0	550
input_data-47	481	0	30	0	49	1	2	1	33	1	2	0	600
unlabeled_data_pool-47	456	0	13	0	18	1	1	1	20	1	2	0	513
input_data-172	1679	1	105	11	167	2	3	4	121	5	2	0	2100
unlabeled_data_pool-172	399	1	40	6	22	1	1	1	45	5	0	0	521

Data counts belonging to each class given as input and added to the unlabeled data pool in each training cycle where a new type of attack is added to the training dataset are listed in *Table 5.3*. The data counts in the last training cycle are available at the end of the table. The default unlabeled pool size threshold is 500. Because of that, the unlabeled pool size equals approximately 500 in the table for each training cycle. In the first training cycle, 82% of the input data instances are added to the unlabeled data pool since the confidence of prediction given by the system is less than the threshold. However, the percentage decreases to 24% in the last training cycle. The system adds less data instances belonging to the classes learned well.

Table 5.4: Least-confident method's unlabeled data pool and selected data table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
unlabeled_data_pool-1	447	0	12	2	22	0	0	3	50	1	0	0	537
selected_data-1	45	0	5	0	4	0	0	0	46	0	0	0	100
unlabeled_data_pool-5	449	4	8	0	10	1	1	1	25	2	0	0	501
selected_data-5	70	0	4	0	9	1	0	0	16	0	0	0	100
unlabeled_data_pool-6	522	1	12	2	16	0	0	2	21	5	0	0	581
selected_data-6	74	0	0	0	9	0	0	1	16	0	0	0	100
unlabeled_data_pool-19	422	2	15	0	13	1	0	3	45	3	1	0	505
selected_data-19	54	1	10	0	8	0	0	0	27	0	0	0	100
unlabeled_data_pool-25	437	1	17	0	17	1	2	0	22	6	0	0	503
selected_data-25	97	0	0	0	1	0	1	0	1	0	0	0	100
unlabeled_data_pool-31	460	1	11	3	14	0	0	3	23	2	0	0	517
selected_data-31	79	1	0	1	3	0	0	2	14	0	0	0	100
unlabeled_data_pool-45	494	0	7	3	10	1	1	3	28	3	0	0	550
selected_data-45	69	0	1	1	4	0	0	3	21	1	0	0	100
unlabeled_data_pool-47	456	0	13	0	18	1	1	1	20	1	2	0	513
selected_data-47	93	0	0	0	4	0	1	0	0	0	2	0	100
unlabeled_data_pool-172	399	1	40	6	22	1	1	1	45	5	0	0	521
selected_data-172	71	0	9	1	2	0	0	0	13	4	0	0	100

Data counts belonging to each class added to the unlabeled data pool and selected by the active learner in each train cycle where a new type of attack is added to the training dataset are listed in *Table 5.4*. The table shows the selection behavior of the active learner based on the least-confident strategy. The default selection parameter value is 100. As shown in the table, 100 data instances were selected by the active learner from the unlabeled data pool in each learning cycle. Least-confident strategy selects the data instances which have the lowest prediction confidence. We can observe that data instances belonging to a new type of class are selected incrementally. Moreover,

the active learner continues to select from data instances belonging to known classes as shown in the table. Until the last cycle, most of the selected data belongs to the BENIGN class. In fact, the unlabeled data pool consists of a high proportion of BENIGN data. Nevertheless, the learner also selects from the data instances belonging to attack classes. The number of new classes selected by the active learner in total is 8.

Table 5.5: Least-confident method's train data table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
train_data-0	9115	0	494	0	0	0	0	0	616	0	0	0	10225
train_data-1	9160	0	499	0	4	0	0	0	662	0	0	0	10325
train_data-5	9419	0	509	0	43	2	0	0	749	0	0	0	10722
train_data-6	9493	0	509	0	52	2	0	2	765	0	0	0	10823
train_data-19	10429	2	533	0	158	4	0	11	985	0	0	0	12122
train_data-25	10983	2	536	0	166	4	2	12	1018	0	0	0	12723
train_data-31	11490	3	537	2	193	5	5	15	1074	0	0	0	13324
train_data-45	12622	4	553	3	242	5	10	27	1257	2	0	0	14725
train_data-47	12801	4	560	3	251	5	11	28	1257	2	3	0	14925
train_data-172	**19728**	**14**	**1287**	**27**	**2509**	**20**	**74**	**85**	**3542**	**130**	**9**	**0**	**27425**
base train data	**181705**	**157**	**10242**	**823**	**18409**	**440**	**464**	**635**	**12704**	**472**	**121**	**52**	**226224**

Data counts belonging to each class in the training dataset is listed in *Table 5.5* starting from the pre-training cycle. Other train cycles are where the new type of attack is added to the training dataset. The data counts in the last training cycle and for the baseline are available at the end of the table. The pre-training dataset only consists of 3 classes: BENIGN, DDos, and PortScan. However, the training dataset contains almost all of the classes in the last training cycle. Only Web attack XSS class is not added to the training dataset. While the training dataset of the baseline contains 80% of the data instances from the BENIGN class, the training dataset of the classifier in the last training cycle contains 71% of the data instances from the BENIGN class. It means the proportion of the data instances from the attack classes is higher in the dataset of the classifier. In addition, the total data instance count in the training dataset of the classifier is 12.12% of the total data instance count in the training dataset of the baseline. This is a significant labeled data reduction.

Effect of initial data split ratio

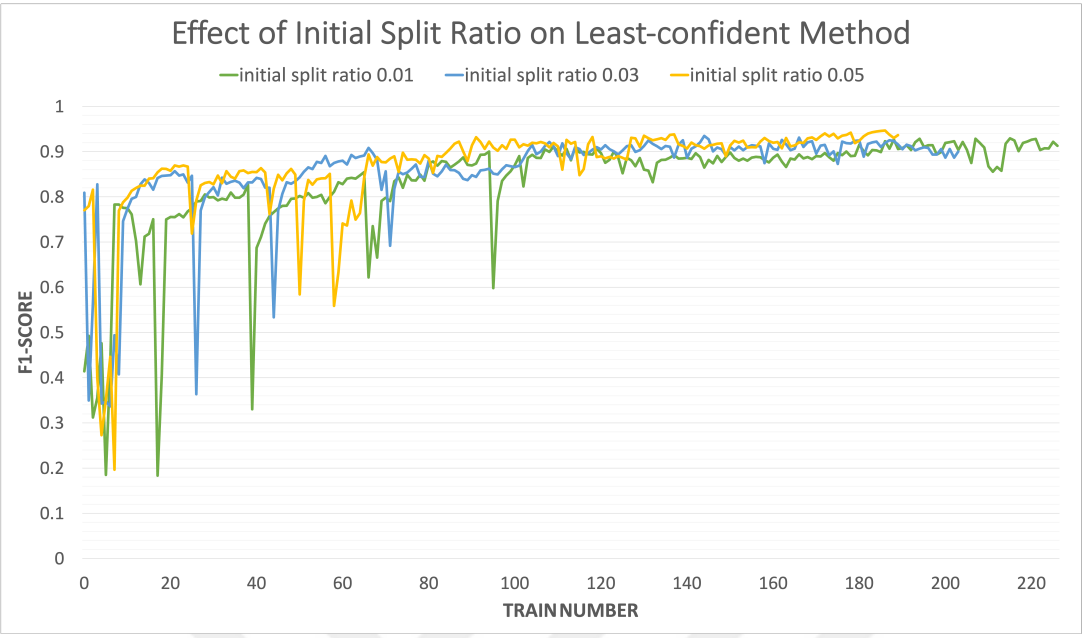


Figure 5.2: Effect of initial split ratio on least confident method

The evaluation graph of the least-confident method for different amounts of pre-training data is shown in *Figure 5.2*. The initial split ratio means the proportion of the data instances reserved from the remaining data after the test data is split for pre-training. As shown in the graph, this parameter doesn't affect the general performance at all. Evolution curves for all of the initial split ratio values have almost the same shape.

Effect of batch size

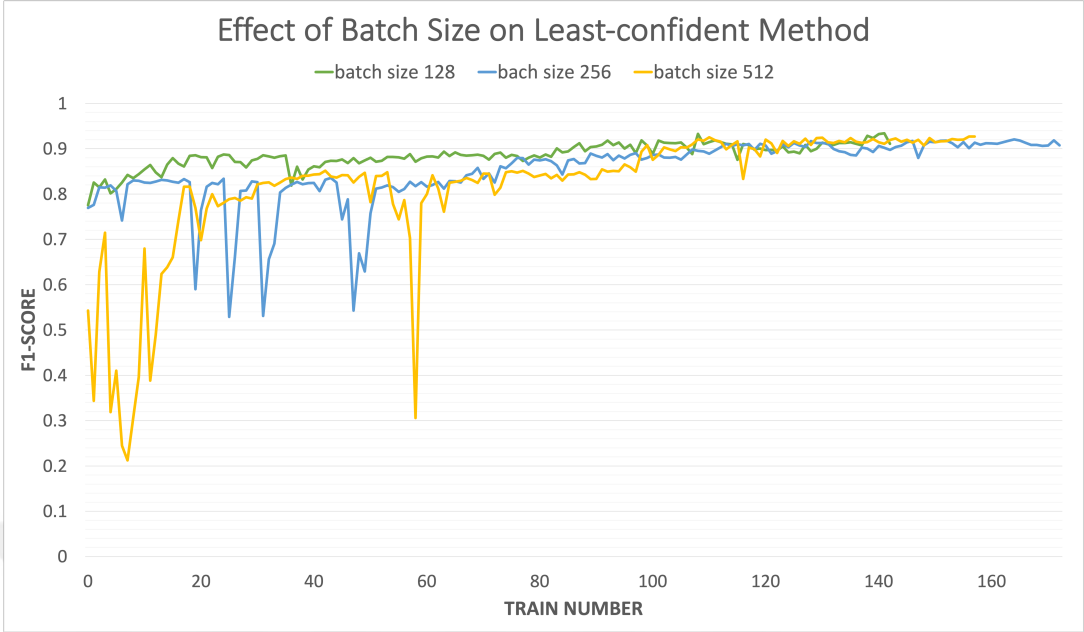


Figure 5.3: Effect of batch size on least confident method

The evaluation graph of the least-confident method for different batch sizes is shown in *Figure 5.3*. When batch size is decreased, we can observe a more smooth evolution curve and faster convergence. ANN learns better with a batch size of 128.

Effect of epochs

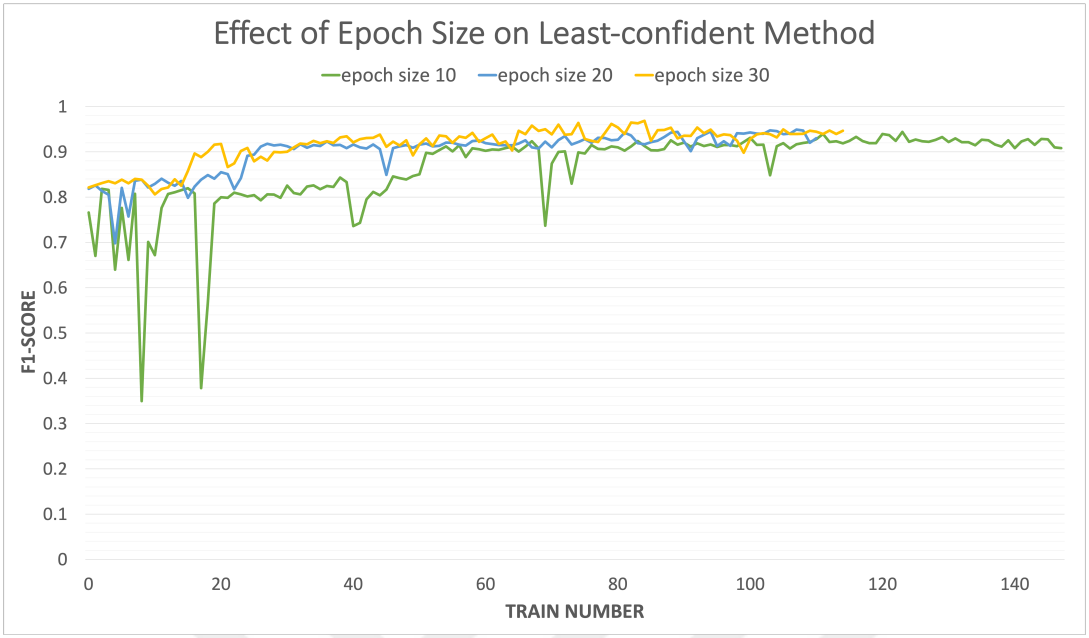


Figure 5.4: Effect of epoch size on least confident method

The evaluation graph of the least-confident method for different epoch sizes is shown in *Figure 5.4*. When epoch size is increased, we can observe a more smooth evolution curve and faster convergence. ANN learns better with an epoch size of 30.

Effect of selection count

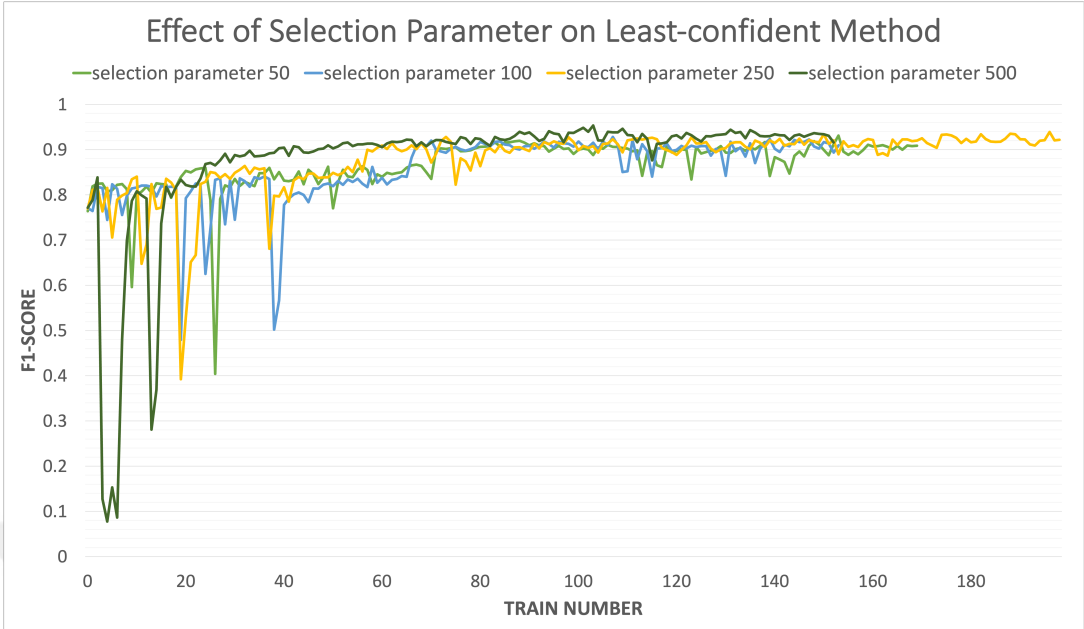


Figure 5.5: Effect of selection parameter on least confident method

The evaluation graph of the least-confident method for different selection parameter values is shown in *Figure 5.5*. The selection parameter means the number of data instances selected from the unlabeled data pool by the active learner. Since the default unlabeled data pool size value is 500, the selection parameter of 500 means selecting all of the data instances in the unlabeled data pool. As shown in the graph, the selection parameter of 500 has huge fluctuations in the first training cycles. The system behavior is approximately the same for all of the other values. Although those values give similar performance, the smallest selection parameter is a better choice because it means using less labeled train data instances.newline

Effect of pool size threshold

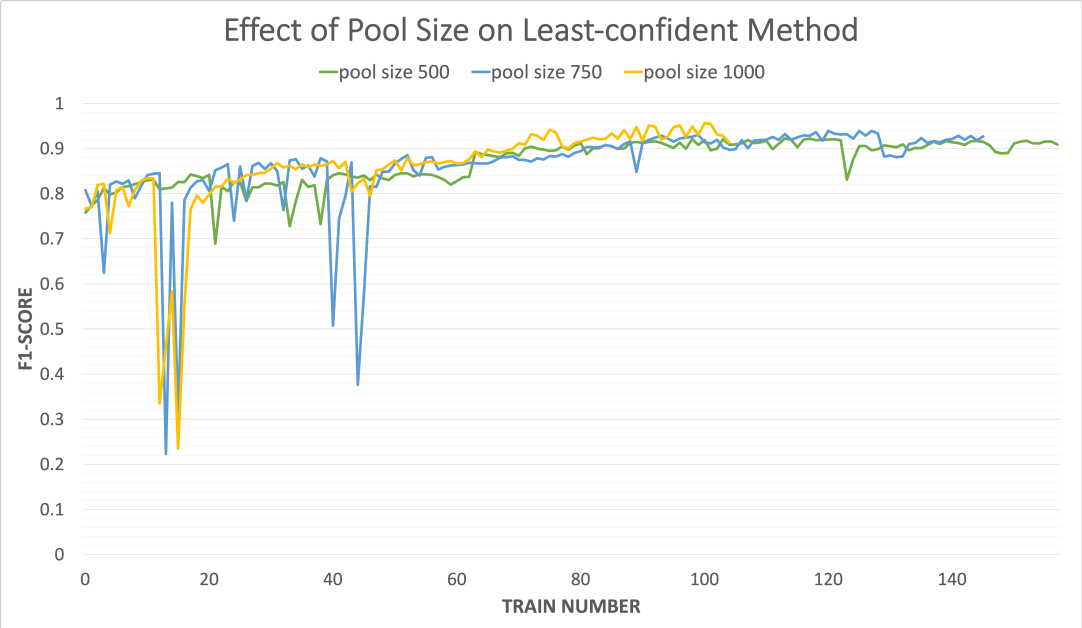


Figure 5.6: Effect of pool size on least confident method

The evaluation graph of the least-confident method for different pool sizes is shown in *Figure 5.6*. For each pool size, the same amount of data instances are selected from the unlabeled data pool since the default value of the selection parameter is 100. Therefore, higher pool size means using fewer training data instances for this experiment. Although the system reaches a similar f1-score value with all of the values, huge fluctuations occur in the evolution curve for the values 750 and 1000. The most suitable pool size value is 500.

Effect of new class min count threshold

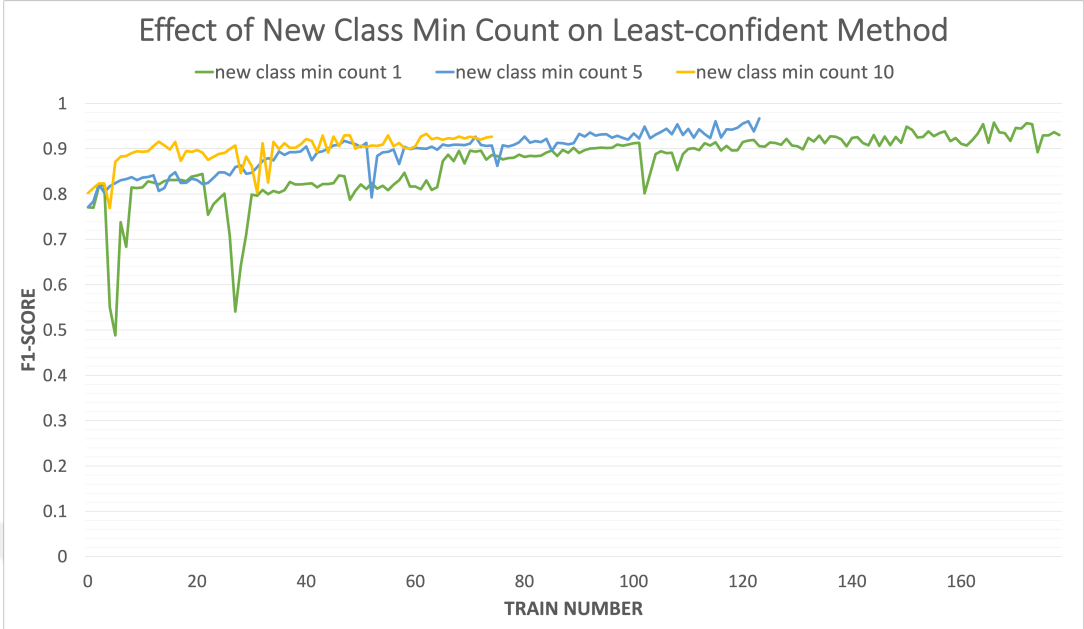


Figure 5.7: Effect of new class min count on least confident method

The evaluation graph of the least-confident method for different new class min count parameter values is shown in *Figure 5.7*. When data instances belonging to a new type of class are selected by the active learner, the system waits until the data instance count for that class reaches a certain value for adding them into the training dataset. The parameter having a value of 1 means no waiting. We observe more performance decreases for the parameter value of 1 when data instances belonging to a new type of class are added to the training dataset as shown in the graph. When the parameter value is 10, fewer new types of classes can be added since some of them are very rare. The performance of the system reaches the highest value with the parameter value of 5.

5.4.2 Margin Strategy Results

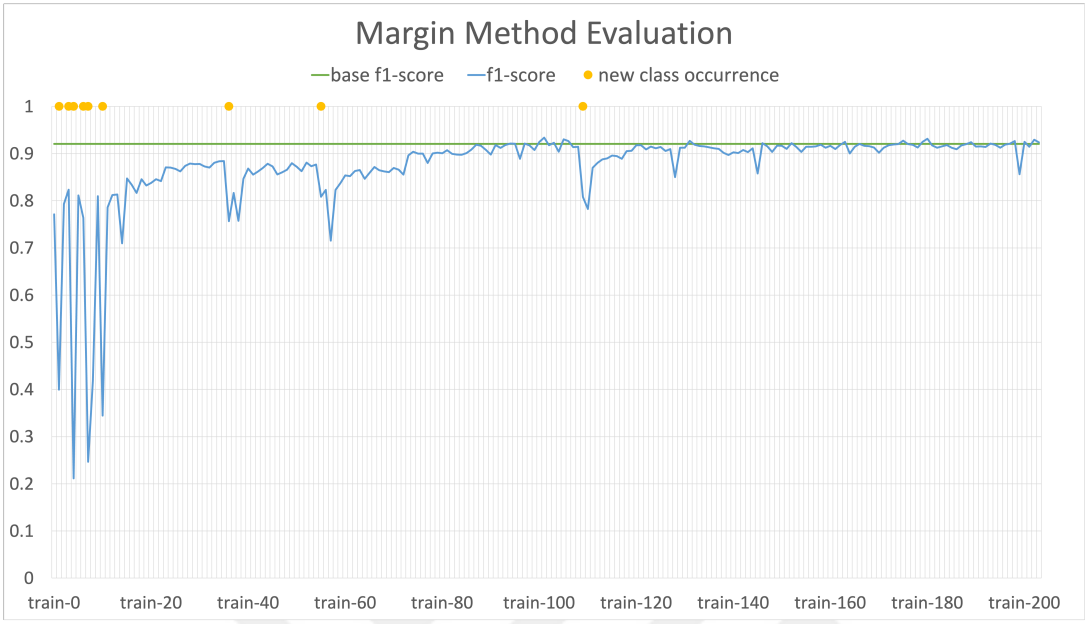


Figure 5.8: Evaluation graph of margin method

Evaluation graph of margin method is shown in *Figure 5.8*. In this graph, we can observe the change in the f1-score of the IDS based on the margin active learning strategy. "train-0" means pre-training cycle, and retraining cycles are named as train-x. In each retraining cycle, new data instances selected by the active learner are added to the training dataset, and the classifier is updated accordingly. Yellow dots show the places where a new type of attack is added to the dataset. A new type of class was added 9 times. As shown in the graph, some amount of performance decrease occurs when a new class is added to the dataset since the new class introduces additional confusion between classes. The system reaches a more stable state after train-20. Although fluctuations occur in the graph, the performance is increased progressively. The green line shows the f1-score of the baseline which is the classifier trained with whole train data instances at once. We can see that the performance of the classifier converges to the performance of the baseline in approximately train-90.

Table 5.6: Margin method's general performance table

name	accuracy	accuracy balanced	f1-score weighted	f1-score macro	precision weighted	precision macro	recall weighted	recall macro
train-0	0.832	0.152	0.771	0.127	0.738	0.137	0.832	0.152
train-1	0.337	0.196	0.399	0.118	0.775	0.158	0.337	0.196
train-3	0.833	0.260	0.823	0.245	0.834	0.261	0.833	0.260
train-4	0.210	0.262	0.212	0.132	0.687	0.160	0.210	0.262
train-6	0.726	0.371	0.764	0.220	0.848	0.210	0.726	0.371
train-7	0.188	0.358	0.247	0.150	0.853	0.214	0.188	0.358
train-10	0.263	0.336	0.345	0.206	0.842	0.235	0.263	0.336
train-36	0.710	0.376	0.757	0.274	0.833	0.244	0.710	0.376
train-55	0.772	0.434	0.808	0.301	0.876	0.280	0.772	0.434
train-109	0.751	0.565	0.808	0.351	0.889	0.339	0.751	0.565
train-203	**0.903**	**0.755**	**0.924**	**0.512**	**0.952**	**0.467**	**0.903**	**0.755**
baseline	**0.913**	**0.620**	**0.920**	**0.527**	**0.934**	**0.560**	**0.913**	**0.620**

Change in all performance measures for the classifier based on margin active learning strategy is listed in *Table 5.6*. There are four performance measures: accuracy, f1-score, precision, and recall. Performance measures have in two types: average and weighted average. Weighted average shows the real use case performance since the performance of the most common classes affects the total performance more. However, the direct average represents the effect of the learning of a new class more clearly. "train-0" means pre-training cycle, and other train cycles are where a new type of attack is added to the dataset. In addition, the performances at the last training cycle and baseline are added to the end of the table. Values of all performance measures are increasing progressively and converging to the baseline as shown in the table. Moreover, the classifier reaches higher values for some of the measures.

Table 5.7: Margin method's f1-score table for each class

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
train-0	0.926	0.000	0.573	0.000	0.000	0.000	0.000	0.000	0.031	0.000	0.000	0.000
train-1	0.427	0.000	0.540	0.000	0.282	0.000	0.000	0.000	0.167	0.000	0.000	0.000
train-3	0.922	0.000	0.541	0.000	0.278	0.578	0.000	0.000	0.617	0.000	0.000	0.000
train-4	0.202	0.000	0.517	0.031	0.189	0.476	0.000	0.000	0.175	0.000	0.000	0.000
train-6	0.859	0.000	0.479	0.098	0.278	0.342	0.000	0.000	0.508	0.074	0.000	0.000
train-7	0.208	0.003	0.484	0.089	0.281	0.119	0.000	0.000	0.608	0.010	0.000	0.000
train-10	0.320	0.006	0.544	0.270	0.415	0.432	0.000	0.000	0.484	0.004	0.000	0.000
train-36	0.831	0.002	0.538	0.468	0.341	0.370	0.004	0.001	0.612	0.124	0.000	0.000
train-55	0.876	0.033	0.541	0.411	0.469	0.493	0.032	0.044	0.700	0.013	0.000	0.000
train-109	0.841	0.038	0.780	0.389	0.716	0.393	0.019	0.084	0.642	0.267	0.021	0.023
train-203	**0.940**	**0.075**	**0.901**	**0.676**	**0.888**	**0.514**	**0.174**	**0.242**	**0.884**	**0.770**	**0.055**	**0.028**
baseline	**0.947**	**0.006**	**0.914**	**0.907**	**0.879**	**0.437**	**0.183**	**0.656**	**0.695**	**0.620**	**0.078**	**0.000**

Changes in f1-scores for each class are listed in *Table 5.7*. "train-0" means pre-training cycle, and other train cycles are where a new type of attack is added to the dataset. In addition, the performances at the last training cycle and baseline are added to the end of the table. We can see that new classes were learned incrementally and all of the classes were learned at the end. In addition, the f1-score of the classifier is similar to the f1-score of the baseline for most classes.

Table 5.8: Margin method's input data and unlabeled data pool table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
input_data-1	475	0	20	3	52	1	0	2	46	1	0	0	600
unlabeled_data_pool-1	416	0	12	2	23	1	0	2	45	1	0	0	502
input_data-3	551	1	33	2	61	3	2	4	41	2	0	0	700
unlabeled_data_pool-3	450	1	7	2	10	3	2	4	41	2	0	0	522
input_data-4	470	1	31	4	43	1	1	3	44	1	1	0	600
unlabeled_data_pool-4	456	1	8	4	13	0	1	3	44	1	1	0	532
input_data-6	495	3	28	2	40	2	1	1	25	2	1	0	600
unlabeled_data_pool-6	482	3	9	2	9	1	1	1	24	2	1	0	535
input_data-7	476	2	35	2	43	0	1	2	36	3	0	0	600
unlabeled_data_pool-7	469	2	27	0	27	0	1	2	36	3	0	0	567
input_data-10	499	0	21	1	41	1	1	2	31	3	0	0	600
unlabeled_data_pool-10	453	0	3	0	13	1	0	2	30	3	0	0	505
input_data-36	956	0	50	4	105	3	3	4	73	1	1	0	1200
unlabeled_data_pool-36	461	0	7	0	34	1	2	4	32	1	1	0	543
input_data-55	821	2	45	8	66	1	0	2	51	3	1	0	1000
unlabeled_data_pool-55	467	1	7	1	14	0	0	2	27	3	1	0	523
input_data-109	801	0	43	6	70	2	0	3	74	0	0	1	1000
unlabeled_data_pool-109	422	0	18	1	48	0	0	3	30	0	0	1	523
input_data-203	1020	0	78	7	104	2	1	2	83	3	0	0	1300
unlabeled_data_pool-203	439	0	22	2	21	0	1	1	34	3	0	0	523

Data counts belonging to each class given as input and added to the unlabeled data pool in each train cycle where a new type of attack is added to the training dataset are listed in *Table 5.8*. The data counts in the last training cycle are available at the end of the table. The default unlabeled pool size threshold is 500. Because of that, the unlabeled pool size equals approximately 500 in the table for each training cycle. In the first training cycle, 83% of the input data instances are added to the unlabeled data pool since the confidence of prediction given by the system is less than the threshold. However, the percentage decreases to 40% in the last training cycle. The system reaches a more stable state progressively and adds fewer data instances to the unlabeled data pool since it learns incrementally.

Table 5.9: Margin method's unlabeled data pool and selected data table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
unlabeled_data_pool-1	416	0	12	2	23	1	0	2	45	1	0	0	502
selected_data-1	42	0	1	0	14	0	0	0	43	0	0	0	100
unlabeled_data_pool-3	450	1	7	2	10	3	2	4	41	2	0	0	522
selected_data-3	54	0	3	0	0	3	0	0	40	0	0	0	100
unlabeled_data_pool-4	456	1	8	4	13	0	1	3	44	1	1	0	532
selected_data-4	71	1	5	2	2	0	0	0	19	0	0	0	100
unlabeled_data_pool-6	482	3	9	2	9	1	1	1	24	2	1	0	535
selected_data-6	82	0	3	0	3	0	0	1	9	2	0	0	100
unlabeled_data_pool-7	469	2	27	0	27	0	1	2	36	3	0	0	567
selected_data-7	75	2	1	0	3	0	0	0	16	3	0	0	100
unlabeled_data_pool-10	453	0	3	0	13	1	0	2	30	3	0	0	505
selected_data-10	79	0	1	0	7	0	0	1	12	0	0	0	100
unlabeled_data_pool-36	461	0	7	0	34	1	2	4	32	1	1	0	543
selected_data-36	56	0	4	0	11	0	2	2	25	0	0	0	100
unlabeled_data_pool-55	467	1	7	1	14	0	0	2	27	3	1	0	523
selected_data-55	72	1	3	0	4	0	0	2	15	2	1	0	100
unlabeled_data_pool-109	422	0	18	1	48	0	0	3	30	0	0	1	523
selected_data-109	71	0	10	1	11	0	0	1	5	0	0	1	100
unlabeled_data_pool-203	439	0	22	2	21	0	1	1	34	3	0	0	523
selected_data-203	76	0	0	0	1	0	0	0	23	0	0	0	100

Data counts belonging to each class added to the unlabeled data pool and selected by the active learner in each train cycle where a new type of attack is added to the training dataset are listed in *Table 5.9*. The table shows the selection behavior of the active learner based on the margin strategy. The default selection parameter value is 100. As shown in the table, 100 data instances were selected by the active learner from the unlabeled data pool in each learning cycle. Margin strategy selects the data instances in which the difference between the most probable and second probable prediction confidence is less. We can observe that data instances belonging to a new type of class are selected incrementally. Moreover, the active learner continues to select from data instances belonging to known classes as shown in the table. Until the last cycle, most of the selected data belongs to the BENIGN class in general. In fact, the unlabeled data pool consists of a high proportion of BENIGN data. Nevertheless, the learner also selects from the data instances belonging to attack classes. The number of new classes selected by the active learner in total is 9.

Table 5.10: Margin method's train data table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
train_data-0	9115	0	494	0	0	0	0	0	616	0	0	0	10225
train_data-1	9157	0	495	0	14	0	0	0	659	0	0	0	10325
train_data-3	9311	0	498	0	14	3	0	0	699	0	0	0	10525
train_data-4	9382	0	503	2	16	3	0	0	718	0	0	0	10624
train_data-6	9563	0	506	3	19	3	0	0	727	2	0	0	10823
train_data-7	9638	3	507	3	22	3	0	0	743	5	0	0	10924
train_data-10	9905	3	508	3	30	3	0	2	766	5	0	0	11225
train_data-36	12014	11	541	4	155	5	3	17	1064	11	0	0	13825
train_data-55	13443	13	581	7	214	5	5	57	1374	24	2	0	15725
train_data-109	17089	15	890	17	646	8	30	108	2249	65	6	2	21125
train_data-203	**23145**	**20**	**1457**	**24**	**1126**	**19**	**67**	**132**	**4377**	**137**	**14**	**7**	**30525**
base train data	**181705**	**157**	**10242**	**823**	**18409**	**440**	**464**	**635**	**12704**	**472**	**121**	**52**	**226224**

Data counts belongs to each class in the training dataset is listed in *Table 5.10* starting from the pre-training cycle. Other train cycles are where the new type of attack is added to the training dataset. The data counts in the last training cycle and for the baseline are available at the end of the table. The pre-training dataset only consists of 3 classes: BENIGN, DDos, and PortScan. However, the training dataset contains all of the classes in the last training cycle. While the training dataset of the baseline contains 80% of the data instances from the BENIGN class, the training dataset of the classifier in the last training cycle contains 75% of the data instances from the BENIGN class. It means the proportion of the data instances from the attack classes is higher in the training dataset of the classifier. In addition, the total data instance count in the training dataset of the classifier is 13.49% of the total data instance count in the training dataset of the baseline. This is a promising labeled data reduction.

Effect of initial data split ratio

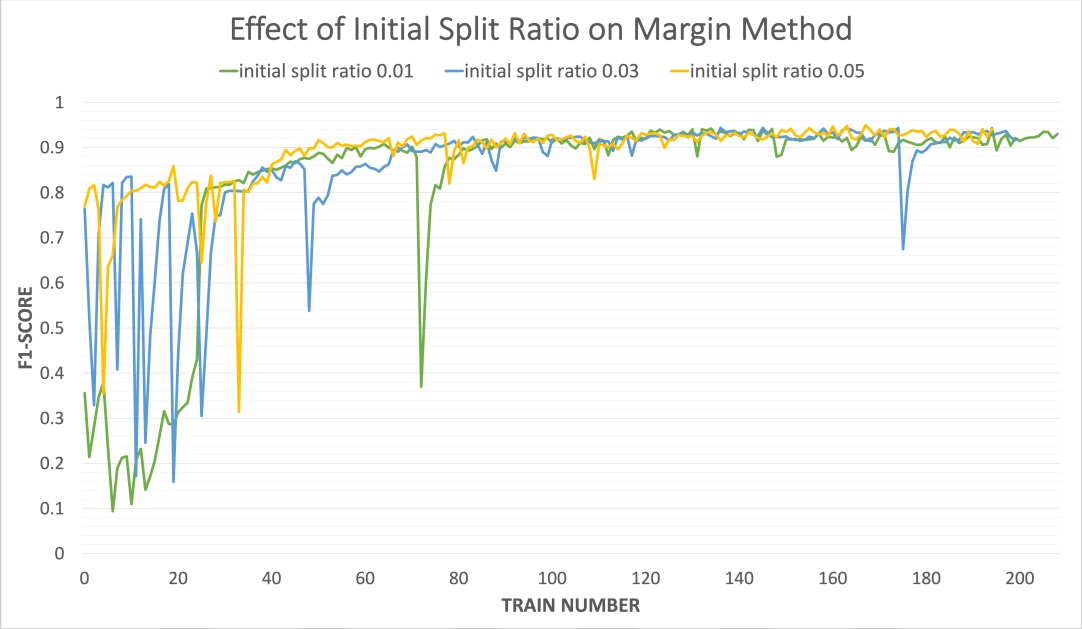


Figure 5.9: Effect of initial split ratio on margin method

Evaluation graph of margin method for different amounts of pre-training data is shown in *Figure 5.9*. The initial split ratio means the proportion of the data instances reserved from the remaining data after the test data is split for pre-training. After train-80, the shape of the curve is almost the same for all of the values. However, the system has too low performance with an initial split ratio of 0.01 through the first 30 train cycles. In addition, lots of large performance decreases occur for the parameter of 0.03 until train 40.

Effect of batch size

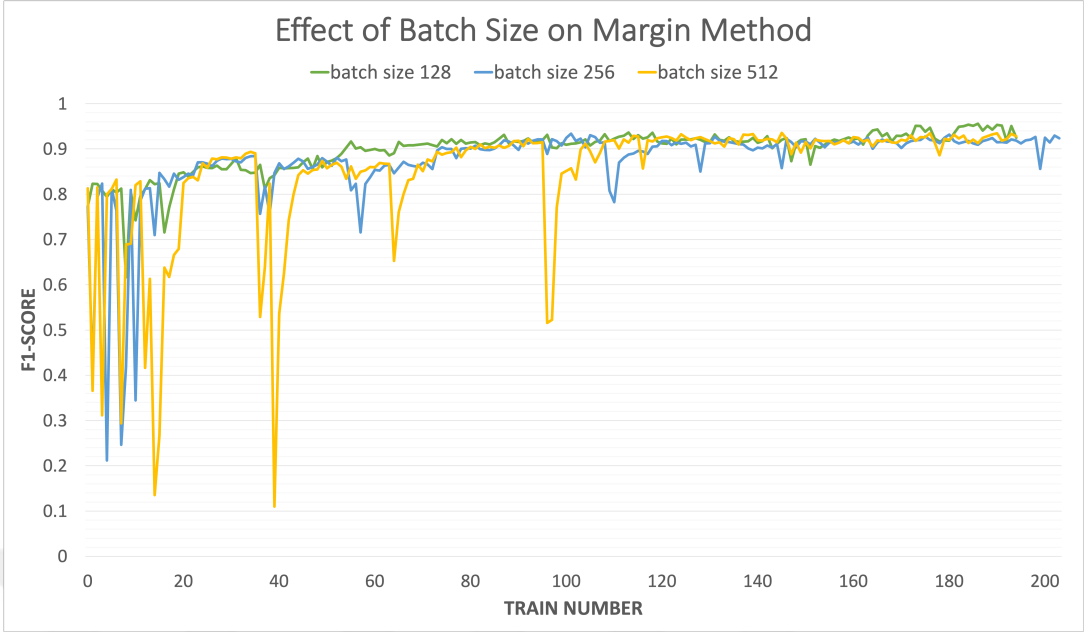


Figure 5.10: Effect of batch size on margin method

The evaluation graph of the margin method for different batch sizes is shown in *Figure 5.10*. Huge fluctuations occur in the evolution curve for the values 256 and 512. ANN converges faster and more smoothly for batch size of 128.

Effect of epochs

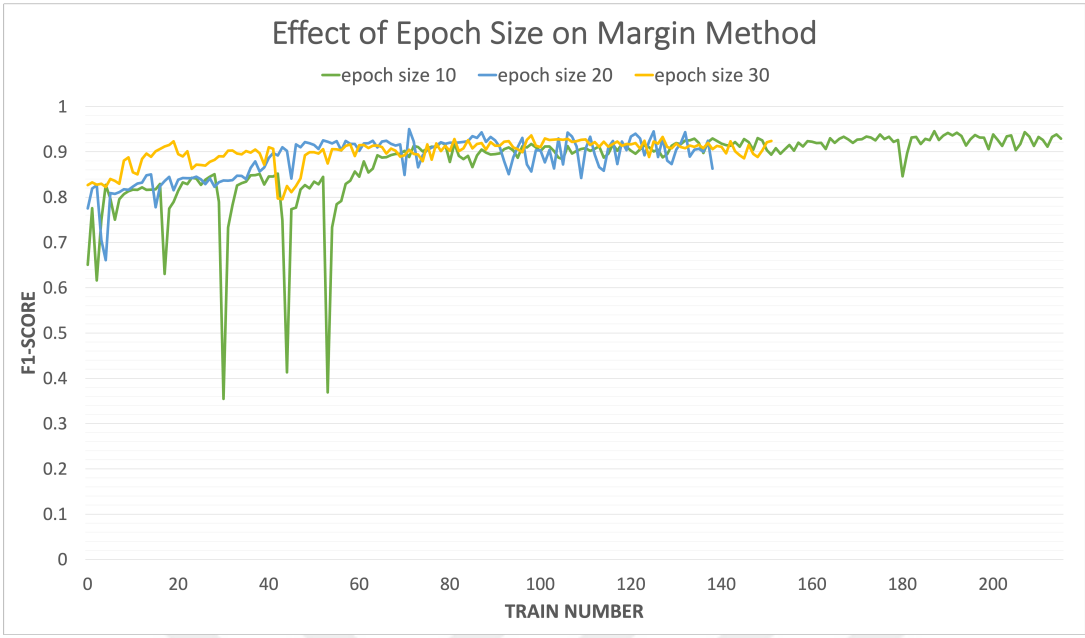


Figure 5.11: Effect of epoch size on margin method

The evaluation graph of the margin method for different epoch sizes is shown in *Figure 5.11*. When epoch size is increased, we can observe a more smooth evolution curve and faster convergence. ANN learns better with a epoch size of 30.

Effect of selection count

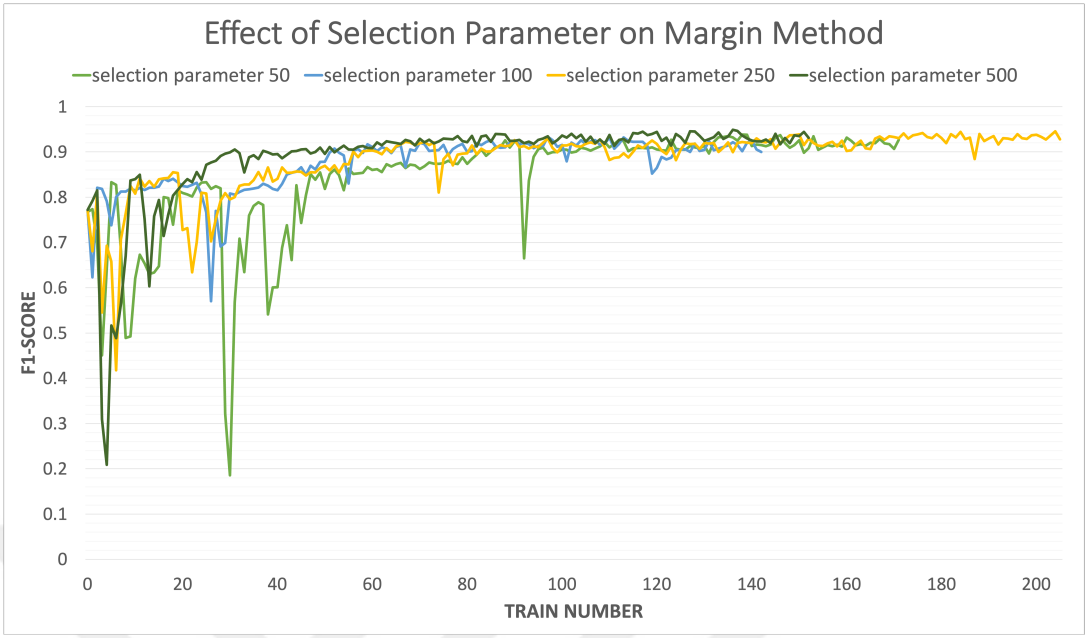


Figure 5.12: Effect of selection parameter on margin method

The evaluation graph of the margin method for different selection parameter values is shown in *Figure 5.12*. The selection parameter means the number of data instances selected from the unlabeled data pool by the active learner. Since the default unlabeled data pool size value is 500, the selection parameter of 500 means selecting all of the data instances in the unlabeled data pool. The system converges to approximately the same performance at the end for all values. However, the convergence is fastest for the value 500 and slowest for the value 50. Huge fluctuations occur for the values 50 and 500 in the first training cycles. The system behavior is almost the same for the values 100 and 250. Although those values give similar performance, the smallest selection parameter is a better choice because it means using less labeled train data instances.

Effect of pool size threshold

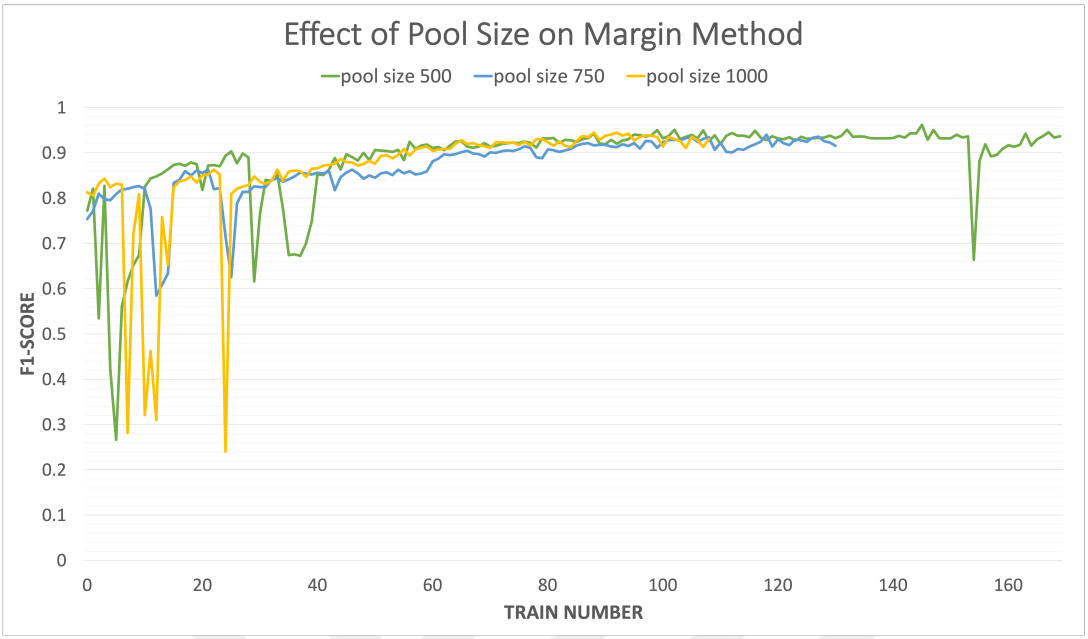


Figure 5.13: Effect of pool size on margin method

The evaluation graph of the margin method for different pool sizes is shown in *Figure 5.13*. For each pool size, the same amount of data instances are selected from the unlabeled data pool since the default value of the selection parameter is 100. Therefore, higher pool size means using fewer train data instances for this experiment. Although the system behavior is similar for all of the values, less performance decreases occur for the value 750 when a new type of class is added to the dataset.

Effect of new class min count threshold

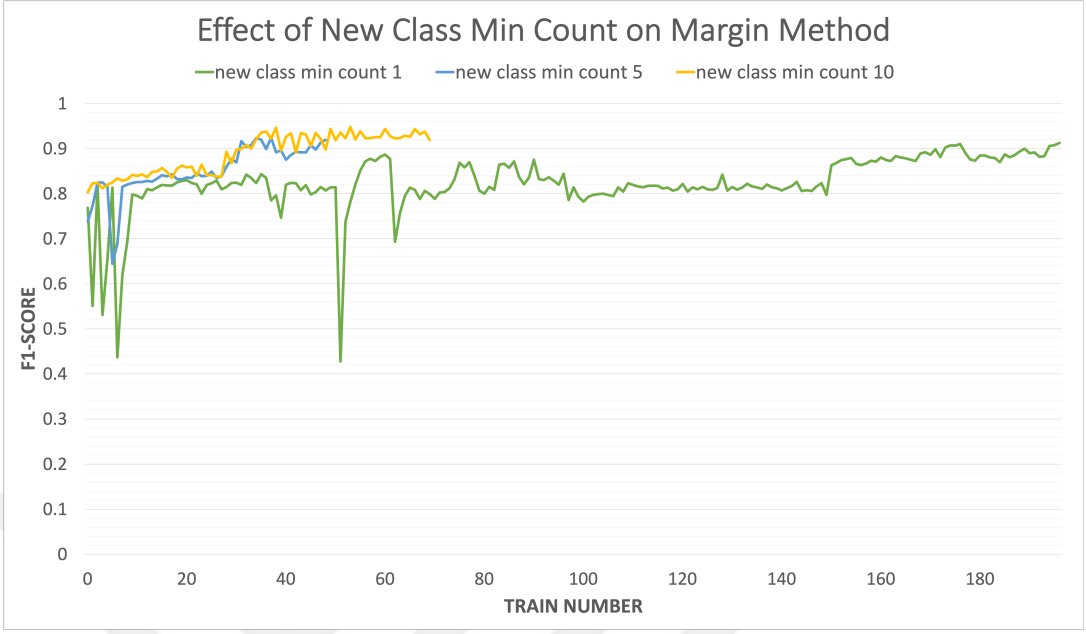


Figure 5.14: Effect of new class min count on margin method

The evaluation graph of the margin method for different new class min count parameter values is shown in *Figure 5.14*. When data instances belonging to a new type of class are selected by the active learner, the system waits until the data instance count for that class reaches a certain value for adding them into the training dataset. The parameter having a value of 1 means no waiting. The system converges faster and has a smoother evolution curve for the values 5 and 10. However, fewer new types of classes can be added since some of them are very rare with those values. Although the value 1 causes a lot of fluctuations, the system can learn more with it. At the end, all of the curves reach the same performance as shown in the graph.

5.4.3 Entropy Strategy Results

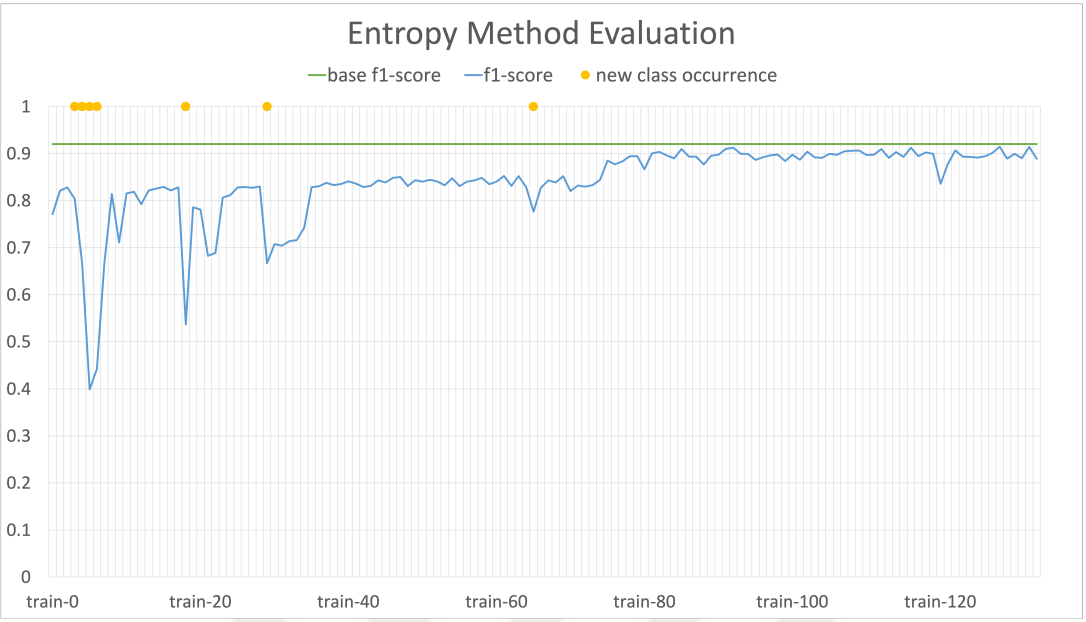


Figure 5.15: Evaluation graph of entropy method

The evaluation graph of the entropy method is shown in *Figure 5.15*. In this graph, we can observe that the change in the f1-score of the IDS is based on entropy active learning strategy. "train-0" means pre-training cycle, and retraining cycles are named as train-x. In each retraining cycle, new data instances selected by the active learner are added to the training dataset, and the classifier is updated accordingly. Yellow dots show the places where a new type of attack is added to the dataset. A new type of class was added 7 times. As shown in the graph, some amount of performance decrease occurs when a new class is added to the dataset since the new class introduces additional confusion between classes. The system reaches a more stable state after train-40. Although fluctuations occur in the graph, the performance is increased progressively. The green line shows the f1-score of the baseline which is the classifier trained with whole train data instances at once. We can see that the performance of the classifier converges to the performance of the baseline in approximately train-90.

Table 5.11: Entropy method's general performance table

name	accuracy	accuracy balanced	f1-score weighted	f1-score macro	precision weighted	precision macro	recall weighted	recall macro
train-0	0.832	0.150	0.771	0.127	0.745	0.148	0.832	0.150
train-3	0.830	0.211	0.804	0.178	0.792	0.168	0.830	0.211
train-4	0.596	0.206	0.665	0.173	0.793	0.167	0.596	0.206
train-5	0.325	0.190	0.399	0.131	0.737	0.141	0.325	0.190
train-6	0.346	0.243	0.442	0.145	0.768	0.168	0.346	0.243
train-18	0.469	0.244	0.537	0.121	0.775	0.123	0.469	0.244
train-29	0.591	0.262	0.667	0.209	0.825	0.194	0.591	0.262
train-65	0.762	0.297	0.777	0.232	0.834	0.245	0.762	0.297
train-133	**0.896**	**0.410**	**0.889**	**0.378**	**0.895**	**0.411**	**0.896**	**0.410**
baseline	**0.913**	**0.620**	**0.920**	**0.527**	**0.934**	**0.560**	**0.913**	**0.620**

Change in all performance measures for the classifier based on entropy active learning strategy is listed in *Table 5.11*. There are four performance measures: accuracy, f1-score, precision, and recall. Performance measures have in two types: average and weighted average. Weighted average shows the real use case performance since the performance of the most common classes affects the total performance more. However, the direct average represents the effect of the learning of a new class more clearly. "train-0" means pre-training cycle, and other train cycles are where a new type of attack is added to the dataset. In addition, the performances at the last training cycle and baseline are added to the end of the table. Values of all performance measures are increasing progressively and converging to the baseline as shown in the table, but the classifier has a lower value in all of the measures.

Table 5.12: Entropy method's f1-score table for each class

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
train-0	0.925	0.000	0.557	0.000	0.000	0.000	0.000	0.000	0.046	0.000	0.000	0.000
train-3	0.922	0.000	0.522	0.000	0.016	0.000	0.000	0.000	0.674	0.000	0.000	0.000
train-4	0.741	0.000	0.570	0.000	0.072	0.000	0.000	0.000	0.690	0.000	0.000	0.000
train-5	0.425	0.021	0.539	0.000	0.037	0.000	0.000	0.000	0.545	0.000	0.000	0.000
train-6	0.472	0.011	0.515	0.000	0.040	0.000	0.000	0.000	0.648	0.057	0.000	0.000
train-18	0.622	0.060	0.500	0.000	0.001	0.000	0.000	0.000	0.260	0.013	0.000	0.000
train-29	0.738	0.337	0.601	0.000	0.062	0.000	0.000	0.001	0.745	0.026	0.000	0.000
train-65	0.871	0.000	0.603	0.000	0.126	0.035	0.000	0.374	0.687	0.087	0.000	0.000
train-133	**0.941**	**0.209**	**0.912**	**0.710**	**0.640**	**0.098**	**0.000**	**0.000**	**0.639**	**0.388**	**0.000**	**0.000**
baseline	**0.947**	**0.006**	**0.914**	**0.907**	**0.879**	**0.437**	**0.183**	**0.656**	**0.695**	**0.620**	**0.078**	**0.000**

Changes in f1-scores for each class are listed in *Table 5.12*. "train-0" means pre-training cycle, and other train cycles are where a new type of attack is added to the dataset. In addition, the performances at the last training cycle and baseline are added to the end of the table. We can see that new classes were learned incrementally and most of the classes were learned at the end. In addition, the f1-score of the classifier is similar to the f1-score of the baseline for some of the classes. On the other hand, there are too many differences between some of them. The classifier performs better just for the BOT class.

Table 5.13: Entropy method's input data and unlabeled data pool table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
input_data-3	1190	2	79	7	126	3	1	8	80	3	1	0	1500
unlabeled_data_pool-3	423	2	14	0	21	0	0	3	42	0	0	0	505
input_data-4	573	3	30	3	55	2	1	1	29	2	1	0	700
unlabeled_data_pool-4	508	3	8	3	20	1	1	1	29	2	1	0	577
input_data-5	476	2	35	2	43	0	1	2	36	3	0	0	600
unlabeled_data_pool-5	435	2	15	2	12	0	0	2	36	3	0	0	507
input_data-6	495	1	20	1	44	1	0	1	34	3	0	0	600
unlabeled_data_pool-6	447	1	6	1	24	1	0	1	32	3	0	0	516
input_data-18	963	0	52	5	106	2	5	6	58	3	0	0	1200
unlabeled_data_pool-18	466	0	3	2	22	1	0	3	29	3	0	0	529
input_data-29	1131	0	51	2	125	1	3	2	82	3	0	0	1400
unlabeled_data_pool-29	446	0	19	2	26	0	2	0	37	0	0	0	532
input_data-65	2951	4	173	18	295	8	10	10	221	7	3	0	3700
unlabeled_data_pool-65	279	0	50	2	73	0	1	2	93	2	0	0	502
input_data-133	1610	0	99	13	149	5	3	5	112	3	1	0	2000
unlabeled_data_pool-133	407	0	11	1	47	1	0	3	41	0	0	0	511

Data counts belonging to each class given as input and added to the unlabeled data pool in each train cycle where a new type of attack is added to the training dataset are listed in *Table 5.13*. The data counts in the last training cycle are available at the end of the table. The default unlabeled pool size threshold is 500. Because of that, the unlabeled pool size equals approximately 500 in the table for each training cycle. The system reaches a more stable state progressively and adds fewer data instances to the unlabeled data pool since it learns incrementally.

Table 5.14: Entropy method's unlabeled data pool and selected data table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
unlabeled_data_pool-3	423	2	14	0	21	0	0	3	42	0	0	0	505
selected_data-3	50	0	9	0	1	0	0	0	40	0	0	0	100
unlabeled_data_pool-4	508	3	8	3	20	1	1	1	29	2	1	0	577
selected_data-4	76	2	5	0	7	0	0	1	9	0	0	0	100
unlabeled_data_pool-5	435	2	15	2	12	0	0	2	36	3	0	0	507
selected_data-5	76	1	3	0	0	0	0	0	17	3	0	0	100
unlabeled_data_pool-6	447	1	6	1	24	1	0	1	32	3	0	0	516
selected_data-6	85	0	1	0	1	0	0	1	10	2	0	0	100
unlabeled_data_pool-18	466	0	3	2	22	1	0	3	29	3	0	0	529
selected_data-18	81	0	1	0	11	1	0	2	4	0	0	0	100
unlabeled_data_pool-29	446	0	19	2	26	0	2	0	37	0	0	0	532
selected_data-29	87	0	0	0	7	0	2	0	4	0	0	0	100
unlabeled_data_pool-65	279	0	50	2	73	0	1	2	93	2	0	0	502
selected_data-65	29	0	16	1	21	0	0	0	31	2	0	0	100
unlabeled_data_pool-133	407	0	11	1	47	1	0	3	41	0	0	0	511
selected_data-133	45	0	5	0	36	1	0	0	13	0	0	0	100

Data counts belonging to each class added to the unlabeled data pool and selected by the active learner in each train cycle where a new type of attack is added to the training dataset are listed in *Table 5.14*. The table shows the selection behavior of the active learner based on the entropy strategy. The default selection parameter value is 100. As shown in the table, 100 data instances were selected by the active learner from the unlabeled data pool in each learning cycle. We can observe that data instances belonging to a new type of class are selected incrementally. Moreover, the active learner continues to select from data instances belonging to known classes as shown in the table. At least 45% of the selected data belongs to the BENIGN class. The number of new classes selected by the active learner in total is 7.

Table 5.15: Entropy method's train data table

name	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total
train_data-0	9115	0	494	0	0	0	0	0	616	0	0	0	10225
train_data-3	9262	0	520	0	2	0	0	0	741	0	0	0	10525
train_data-4	9338	2	525	0	9	0	0	0	750	0	0	0	10624
train_data-5	9414	3	528	0	9	0	0	0	767	3	0	0	10724
train_data-6	9499	3	529	0	10	0	0	2	777	5	0	0	10825
train_data-18	10484	3	539	0	63	2	0	10	916	8	0	0	12025
train_data-29	11482	3	555	0	104	2	3	16	950	9	0	0	13124
train_data-65	14395	4	626	2	358	4	11	59	1253	12	0	0	16724
train_data-133	**18962**	**5**	**887**	**29**	**1701**	**10**	**16**	**100**	**1799**	**15**	**0**	**0**	**23524**
base train data	**181705**	**157**	**10242**	**823**	**18409**	**440**	**464**	**635**	**12704**	**472**	**121**	**52**	**226224**

Data counts belonging to each class in the training dataset is listed in *Table 5.15* starting from the pre-training cycle. Other train cycles are where the new type of attack is added to the training dataset. The data counts in the last training cycle and for the baseline are available at the end of the table. The pre-training dataset only consists of 3 classes: BENIGN, DDos, and PortScan. However, the training dataset contains most of the classes in the last training cycle. The training dataset of the baseline and the classifier in the last training cycle contain 80% of the data instances from the BENIGN class. The total data instance count in the training dataset of the classifier is 10.39% of the total data instance count in the training dataset of the baseline. This is a substantial labeled data reduction.

Effect of initial data split ratio

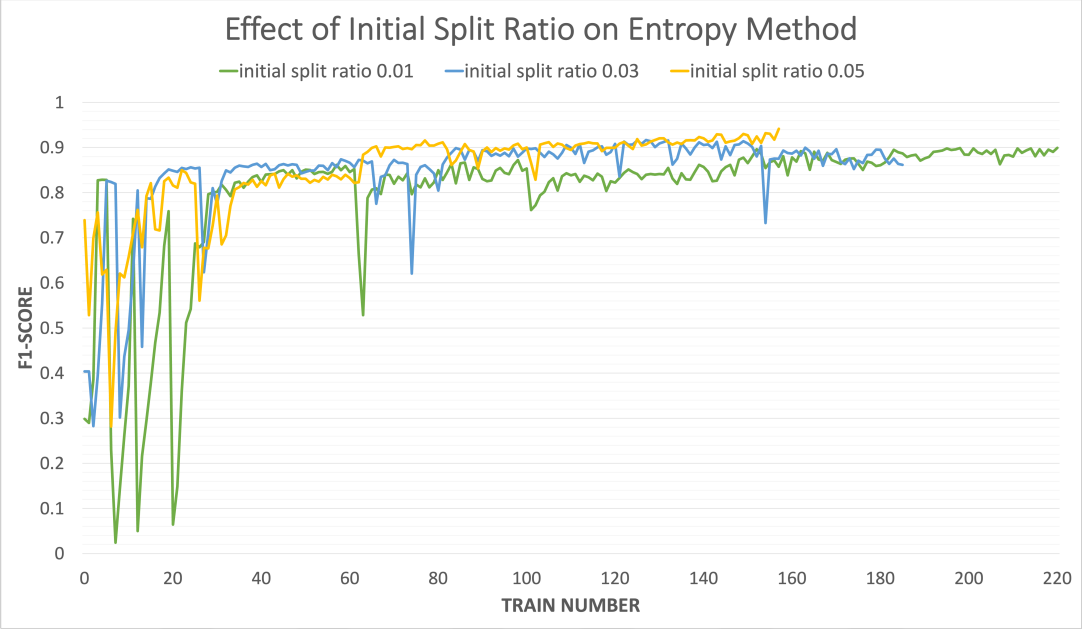


Figure 5.16: Effect of initial split ratio on entropy method

The evaluation graph of the entropy method for different amounts of pre-training data is shown in *Figure 5.16*. The initial split ratio means the proportion of the data instances reserved from the remaining data after the test data is split for pre-training. Lots of large performance decreases occur for the parameter value of 0.01 until train 30. The curve for the parameter value of 0.05 converges to the highest performance value. Moreover, it has fewer fluctuations when compared to curves of the other values. The best initial split ratio value is 0.05.

Effect of batch size

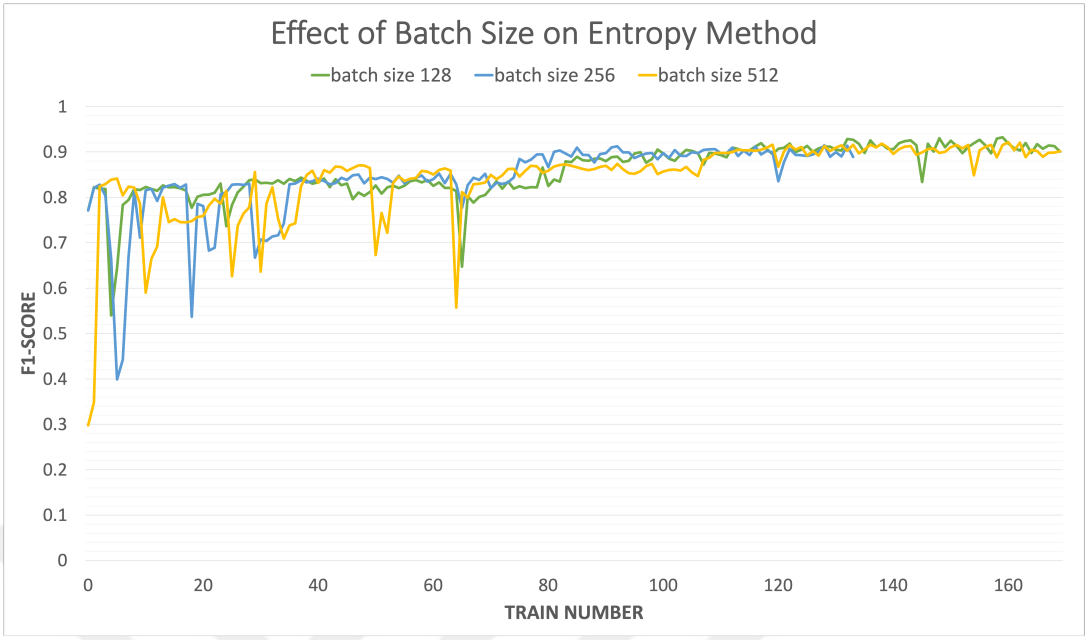


Figure 5.17: Effect of batch size on entropy method

The evaluation graph of the entropy method for different batch sizes is shown in *Figure 5.17*. The performance for all of the values converges to almost the same value, and the shape of the curves are similar. However, we observed that the curve for the parameter value of 128 is the smoothest one.

Effect of epochs

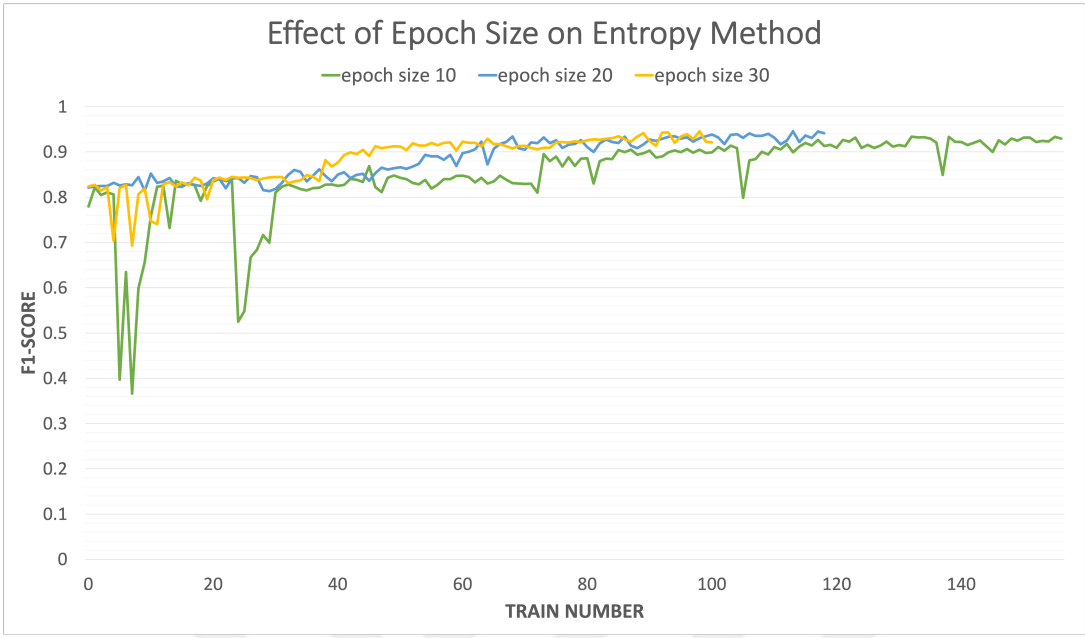


Figure 5.18: Effect of epoch size on entropy method

The evaluation graph of the entropy method for different epoch sizes is shown in *Figure 5.18*. When epoch size is increased, we can observe a more smooth evolution curve and faster convergence. ANN learns better with a epoch size of 30.

Effect of selection count

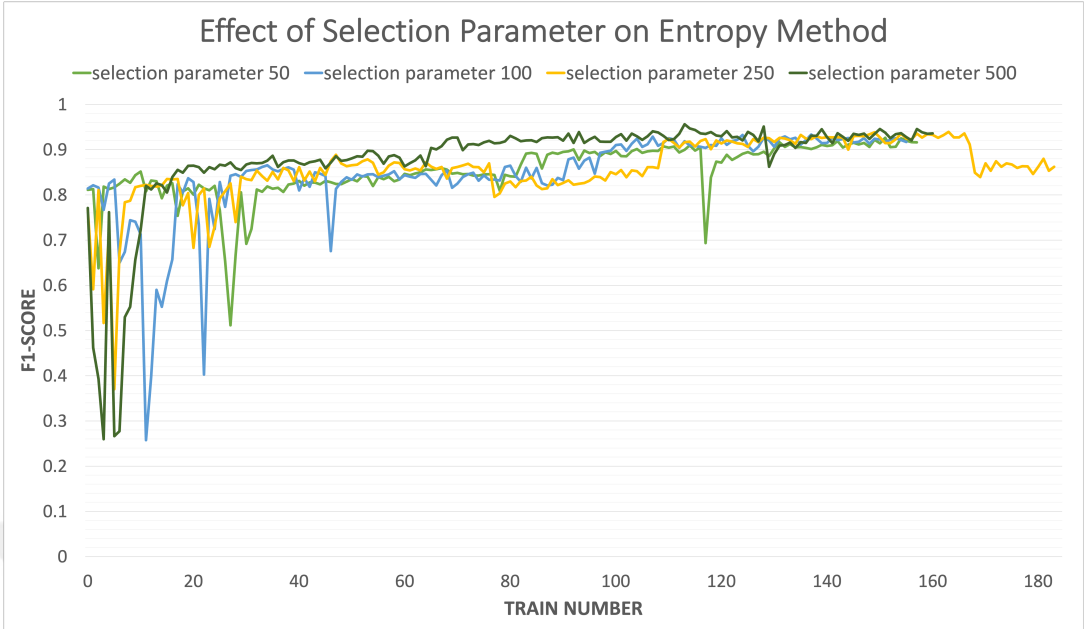


Figure 5.19: Effect of selection parameter on entropy method

The evaluation graph of the entropy method for different selection parameter values is shown in *Figure 5.19*. The selection parameter means the number of data instances selected from the unlabeled data pool by the active learner. Since the default unlabeled data pool size value is 500, the selection parameter of 500 means selecting all of the data instances in the unlabeled data pool. Although the system converges to approximately the same performance for all values, there is a performance decrease at the end for the value 250. The convergence is fastest for the value 500 and almost the same for the other values. Huge fluctuations occur for all of them, especially in the first training cycles.

Effect of pool size threshold

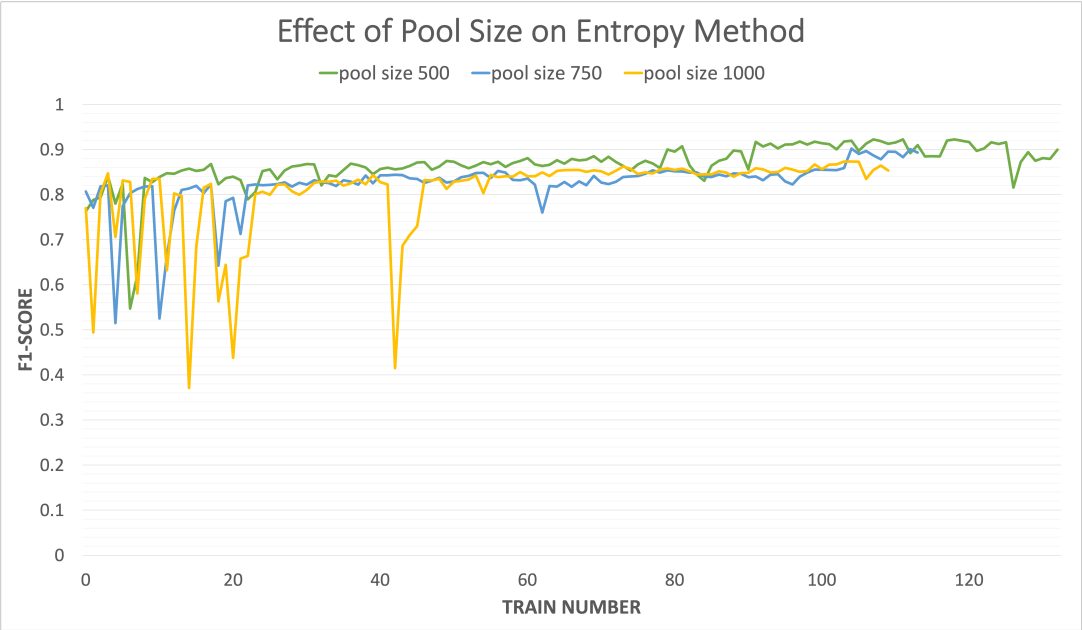


Figure 5.20: Effect of pool size on entropy method

The evaluation graph of the entropy method for different pool sizes is shown in *Figure 5.20*. For each pool size, the same amount of data instances are selected from the unlabeled data pool since the default value of the selection parameter is 100. Therefore, higher pool size means using fewer train data instances for this experiment. The system reaches the highest performance with the value 500. Moreover, the evolution curve is smoothest for it. On the other hand, huge fluctuations occur for the values 750 and 1000.

Effect of new class min count threshold

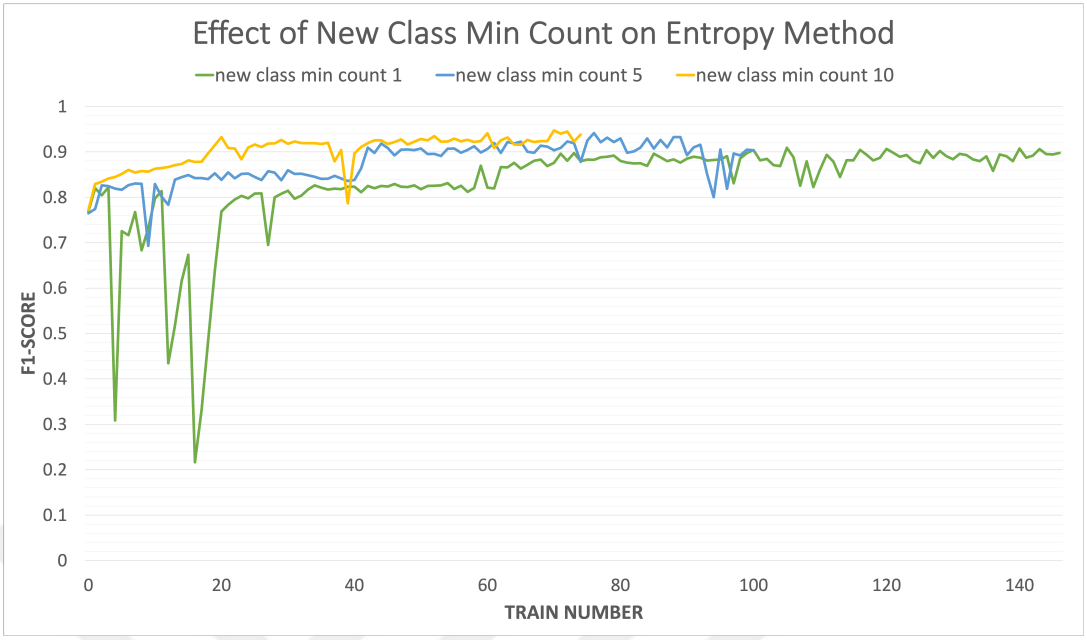


Figure 5.21: Effect of new class min count on entropy method

The evaluation graph of the entropy method for different new class min count parameter values is shown in *Figure 5.21*. When data instances belonging to a new type of class are selected by the active learner, the system waits until the data instance count for that class reaches a certain value for adding them into the training dataset. The parameter having a value of 1 means no waiting. The system converges faster and has a smoother evolution curve while the new class min count parameter decreases. On the other hand, fewer new types of classes can be added since some of them are very rare with those values. Although the value 1 causes a lot of fluctuations, the system can learn more with it.

5.4.4 General Discussion

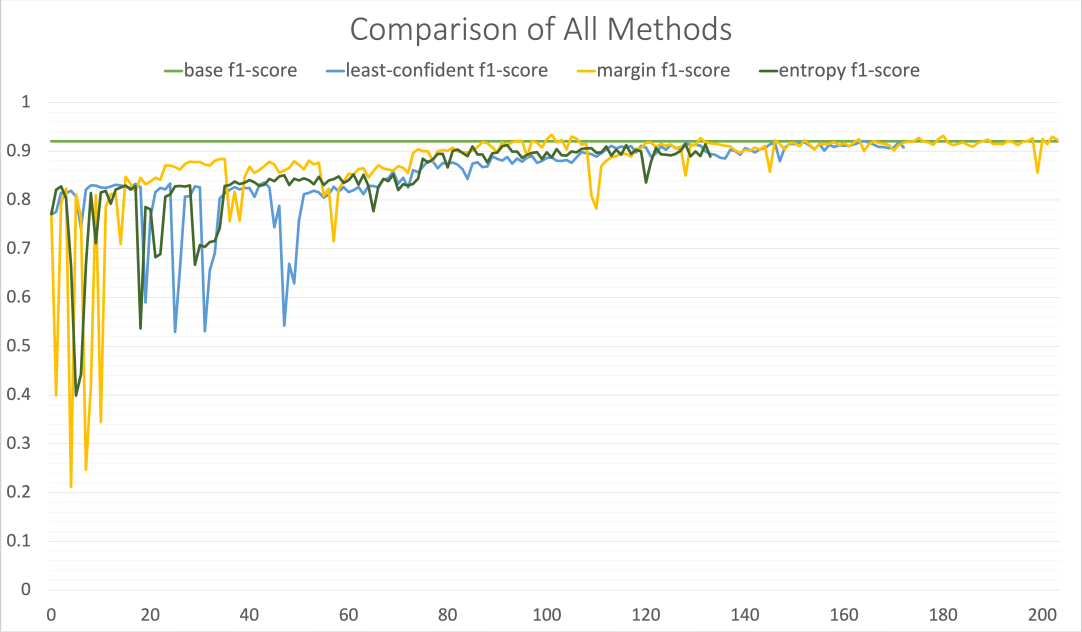


Figure 5.22: Comparison of all methods

The evaluation graph for all of the three methods and the baseline is shown in *Figure 5.22*. The default configuration is used for all of them. As shown in the graph, all of the three methods converge to the baseline at the end, but the margin converges faster than the others.

Table 5.16: All methods general performance comparison table

method	accuracy	accuracy balanced	f1-score weighted	f1-score macro	precision weighted	precision macro	recall weighted	recall macro
least-confident	0.894	0.723	0.908	0.541	0.927	0.530	0.894	0.723
margin	0.903	0.755	0.924	0.512	0.952	0.467	0.903	0.755
entropy	0.896	0.410	0.889	0.378	0.895	0.411	0.896	0.410
baseline	0.913	0.620	0.920	0.527	0.934	0.560	0.913	0.620

For all of the three methods and the baseline, all performance measures are listed in *Table 5.16*. The values are collected from the last training cycle. There are four performance measures: accuracy, f1-score, precision, and recall. Performance measures are in two types: average and weighted average. Weighted average shows the real use case performance since the performance of the most common classes affects the

total performance more. For most of the performance measures, the margin method has a higher value than the other active learning methods. All of the three methods have similar values with the baseline in any weighted version of the performance measures. When we consider the whole table, the margin method is the best and the entropy method is the worse in terms of those measures.

Table 5.17: All methods f1-score comparison table for each class

method	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
least-confident	0.936	0.078	0.904	0.534	0.708	0.534	0.415	0.821	0.887	0.622	0.051	0.000
margin	0.940	0.075	0.901	0.676	0.888	0.514	0.174	0.242	0.884	0.770	0.055	0.028
entropy	0.941	0.209	0.912	0.710	0.640	0.098	0.000	0.000	0.639	0.388	0.000	0.000
baseline	0.947	0.006	0.914	0.907	0.879	0.437	0.183	0.656	0.695	0.620	0.078	0.000

F1-score for each class is listed in *Table 5.17* for all of the three methods and the baseline. Performances for BENIGN, DDos, PortScan, and DoS Hulk classes are high for all methods and baseline since those are the most common classes in the dataset. For the Bot class, all of the three active learning methods perform better than the baseline. On the other hand, the baseline overperforms for DoS Golden Eye. Web attack XSS class can be detected just with the margin method.

Table 5.18: All methods train data comparison table

method	BENIGN	Bot	DDoS	DoS Golden Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS	total	percantage
least-confident	19728	14	1287	27	2509	20	74	85	3542	130	9	0	27425	12.12%
margin	23145	20	1457	24	1126	19	67	132	4377	137	14	7	30525	13.49%
entropy	18962	5	887	29	1701	10	16	100	1799	15	0	0	23524	10.39%
base train data	181705	157	10242	823	18409	440	464	635	12704	472	121	52	226224	100%

Data counts belonging to each class in the training dataset are listed in *Table 5.18* for all of the three methods and the baseline. Train data information for active learning methods is from the last training cycle. Total data instance count in the training datasets and also their percentages are also given in the table. All three method have similar class distributions and uses less than 15% of the total data for training. This is a significant data usage reduction. The entropy method is the best in terms of data

usage. Least-confident is the second and margin is the third. Data instances belonging to web attack brute force and web attack XSS are not selected by some of the active learners. In fact, those classes are rare in the dataset.

CHAPTER 6

CONCLUSION

An increase in usage of the Internet comes with the growth of malware. Therefore, IDS has become a much more critical position in our lives. Its performance depends on adaptability to changing attack types and a substantial amount of labeled data. Unfortunately, labeled data amount for intrusion detection is very limited, and deep learning methods have certain difficulties with learning incrementally. Active learning has great potential to deal with those problems.

In this thesis, we proposed a network intrusion detection system with active learning which can learn new attack types incrementally in an efficient way using a small amount of labeled train data. As far as we know, this is the first time that active learning is applied to the multi-class classification problem of the intrusion detection domain for class incremental learning purposes. An ANN-based classifier is preferred for use in this system. Since uncertainty-based active learning methods are a straightforward way of using them for incremental learning purposes, we worked on three uncertainty-based methods: least-confident, margin, and entropy. We evaluated them in the CICIDS-2017 dataset which can model modern network traffic. The parameters that affect the performance of the proposed IDS were analyzed comprehensively in the experiments. Moreover, the performance of the three methods and their data usage behavior is compared. In addition, their performances are checked against the baseline which is the classifier trained with the whole dataset.

Experiments show that all of the three active learning methods require less than 15% of the total data for training to reach the baseline performance. This is a significant data usage reduction. The entropy method is the best in terms of data usage. The least-confident is the second and the margin is the third. On the other hand, the

margin method has a higher value than the other active learning methods for most of the performance measures and the entropy method is the worst in terms of those measures.

As a future work, the methods that perform better on small train datasets such as few-shot learning can be used as a classifier instead of ANN. It may be possible to achieve higher performance with those methods, especially for the classes that are too rare. Uncertainty-based methods can be replaced with other active learning methods after being modified for using class incremental learning. Moreover, deep active learning methods can be tried to select data instances more wisely. Research papers about those issues are very limited. They are waiting to be explored by researchers.

REFERENCES

[1] "McAfee labs threat report." https://www.mcafee.com/enterprise/en-us/assets/reports/rp-threats-jun-2021.pdf, Jun 2021. [Online; accessed 09-August-2022].

[2] R. Shirey, "RFC2828: Internet security glossary," tech. rep., GTE/BBN Technologies, USA, 2000.

[3] J. P. Anderson, "Computer security threat monitoring and surveillance," tech. rep., James P. Anderson Company, 1980.

[4] R. Kemmerer and G. Vigna, "Intrusion detection: A brief history and overview," *Computer*, vol. 35, pp. 27–30, 2002.

[5] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, pp. 1–22, 2019.

[6] W. Stallings and L. Brown, *Computer security: principles and practice*. Pearson Education, Inc, 2015.

[7] D. H. Hubel, *Eye, brain, and vision*. Scientific American Library/Scientific American Books, 1995.

[8] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton," *Cornell Aeronautical Lab*, 1957.

[9] A. G. Ivakhnenko and V. G. Lapa, "Cybernetic predicting devices," 1965.

[10] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[11] B. Settles, "Active learning literature survey," tech. rep., University of Wisconsin–Madison, 2010.

[12] C. Aggarwal, X. Kong, Q. Gu, J. Han, and P. Yu, *Active learning: A survey*. CRC Press, 2014.

[13] G. Schohn and D. Cohn, "Less is more: Active learning with support vector machines," in *Proceedings of the Seventeenth International Conference on Machine Learning*, p. 839–846, 2000.

[14] D. Lewis and W. Gale, "A sequential algorithm for training text classifiers," in *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1994*, pp. 3–12, 1994.

[15] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 623–656, 10 1948.

[16] H. Seung, M. Opper, and H. Sompolinsky, "Query by committee," in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 287–294, 1992.

[17] B. Settles, M. Craven, and S. Ray, "Multiple-instance active learning," in *Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference*, 2009.

[18] B. Settles and M. Craven, "An analysis of active learning strategies for sequence labeling tasks," in *EMNLP 2008 - 2008 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference: A Meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1070–1079, 2008.

[19] G. Parisi, R. Kemker, J. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[20] R. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, pp. 128–135, 1999.

[21] M. D. Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 3366–3385, 2022.

[22] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. Lampert, "iCaRL: Incremental classifier and representation learning," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 5533–5542, 2017.

[23] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 6468–6477, 2017.

[24] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, pp. 3521–3526, 2017.

[25] A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

[26] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences (Switzerland)*, vol. 9, p. 4396, 2019.

[27] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *Journal of Network and Computer Applications*, vol. 169, p. 102767, 2020.

[28] M. Almgren and E. Jonsson, "Using active learning in intrusion detection," in *Proceedings of the Computer Security Foundations Workshop*, vol. 17, pp. 88–98, 2004.

[29] Y. Li and L. Guo, "An active learning based TCM-KNN algorithm for supervised network intrusion detection," *Computers and Security*, vol. 26, pp. 459–467, 2007.

[30] J. Long, J.-P. Yin, E. Zhu, and W.-T. Zhao, "A novel active cost-sensitive learning method for intrusion detection," in *Proceedings of the 7th International Con-*

ference on Machine Learning and Cybernetics, ICMLC, vol. 2, pp. 1099–1104, 2008.

[31] C. H. Mao, H. M. Lee, D. Parikh, T. Chen, and S. Y. Huang, "Semi-supervised co-training and active learning based approach for multi-view intrusion detection," in *Proceedings of the ACM Symposium on Applied Computing*, pp. 2042–2048, 2009.

[32] N. Seliya and T. Khoshgoftaar, "Active learning with neural networks for intrusion detection," in *2010 IEEE International Conference on Information Reuse and Integration, IRI 2010*, pp. 49–54, 2010.

[33] Y. Meng and L. F. Kwok, *Enhancing false alarm reduction using pool-based active learning in network intrusion detection*, vol. 7863 LNCS. Springer, 2013.

[34] V. Kumari and P. Varma, "A semi-supervised intrusion detection system using active learning svm and fuzzy c-means clustering," in *Proceedings of the International Conference on IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2017*, pp. 481–485, 2017.

[35] K. Yang, J. Ren, Y. Zhu, and W. Zhang, "Active learning for wireless IoT intrusion detection," *IEEE Wireless Communications*, vol. 25, pp. 19–25, 2018.

[36] J. Li, W. Wu, and D. Xue, "An intrusion detection method based on active transfer learning," *Intelligent Data Analysis*, vol. 24, pp. 263–283, 2020.

[37] Y. Gu and D. Zydek, "Active learning for intrusion detection," in *Proceedings - 2014 National Wireless Research Collaboration Symposium: Rapidly Transitioning Wireless Spectrum - Using Research to Deployable Innovations, NWRCS 2014*, pp. 117–122, 2014.

[38] S. McElwee, "Active learning intrusion detection using k-means clustering selection," in *Conference Proceedings - IEEE SOUTHEASTCON*, pp. 1–7, 2017.

[39] S. McElwee and J. Cannady, "Cyber situation awareness with active learning for intrusion detection," in *Conference Proceedings - IEEE SOUTHEASTCON*, vol. 2019-April, pp. 1–7, 2019.

[40] L. Boukela, G. Zhang, M. Yacoub, and S. Bouzefrane, "A near-autonomous and incremental intrusion detection system through active learning of known and unknown attacks," in *Conference Digest - 2021 International Conference on Security, Pattern Analysis, and Cybernetics, SPAC 2021*, pp. 374–379, 2021.

[41] J. Long, W. Zhao, F. Zhu, and Z. Cai, "Active learning to defend poisoning attack against semi-supervised intrusion detection classifier," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 19, pp. 93–106, 2011.

[42] G. Alqaralleh, M. Alshraideh, and A. Alrodan, "A comparison study between different sampling strategies for intrusion detection system of active learning model," *Journal of Computer Science*, vol. 14, pp. 1155–1173, 2018.

[43] Z. Zhang, Y. Zhang, D. Guo, and M. Song, "A scalable network intrusion detection system towards detecting, discovering, and learning unknown attacks," *International Journal of Machine Learning and Cybernetics*, vol. 12, pp. 1649–1665, 2021.

[44] M. Martina and G. Foresti, "A continuous learning approach for real-time network intrusion detection," *International Journal of Neural Systems*, vol. 31, p. 2150060, 2021.

[45] S. Amalapuram, A. Tadwai, R. Vinta, S. Channappayya, and B. Tamma, "Continual learning for anomaly based network intrusion detection," in *2022 14th International Conference on COMmunication Systems and NETworkS, COMSNETS 2022*, pp. 497–505, 2022.

[46] J. Lin, Y. Wei, W. Li, and J. Long, *Intrusion Detection System Based on Deep Neural Network and Incremental Learning for In-Vehicle CAN Networks*, vol. 1557 CCIS. Springer, 2022.

[47] I. Sharafaldin, A. Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, vol. 2018-January, pp. 108–116, 2018.